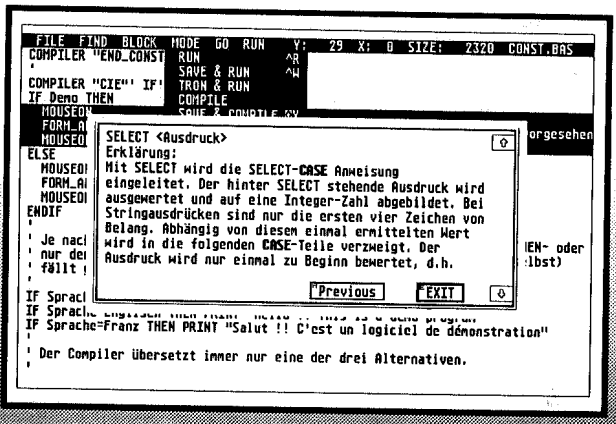
OMIKRON.

INTERPRETER 3.5

ATARI ST



... für den anspruchsvollen Programmierer

Andreas Eberlein, Günther Heidel, Stefan Rinke, Bernhard Sandkühler, Frank Zimmer:
Handbuch zum OMIKRON.BASIC Interpreter 3.5

Auflage: 6 5 4 3 2 (Die letzte Zahl ist jeweils gültig.)
Jahr: 95 94 93 92

Autoren des Programmes OMIKRON.BASIC 3.5 sind:

Artur Södler, Thomas Kemp, Stefan Rinke

Die Programmautoren sind Herrn Dietrich Raisin für viele wertvolle Tips und Anregungen in besonderem Maße zu Dank verpflichtet.

Sponheimstr. 12
D-7530 Pforzheim
Tel. 07231/356033

Haftung und Garantie

Das vorliegende Handbuch und die dazugehörigen Programme wurden mit der größten Sorgfalt erstellt. Trotzdem sind Fehler nie ganz auszuschließen. Daher möchten wir darauf hinweisen, daß wir weder eine Garantie für die Fehlerfreiheit geben noch die Haftung für irgendwelche Folgen, gleich ob durch Fehler im Handbuch, in der Software oder in der Hardware verursacht, übernehmen können. Für die Mitteilung eventueller Fehler im Interpreter oder im Compiler sind wir jederzeit dankbar.

Urheberrecht

Ein Kopieren der OMIKRON.BASIC-Diskettenversion ist nur zu Sicherungszwecken erlaubt. Für Schäden, die dadurch entstehen, daß das Original oder die Sicherungskopie in die Hände dritter gelangt, sind Sie haftbar.

Gegen Raubkopierer werden wir mit gerichtlichen Schritten vorgehen.

OMIKRON. Soft + Hardware GmbH

Vor Inbetriebnahme lesen!

1 Diesem Produkt ist eine Kundenservice-Karte beigefügt. Nur, wenn Sie diese Karte vollständig ausgefüllt an uns senden, können wir Sie als rechtmäßigen Benutzer dieses Programms registrieren.
Sie erhalten nur dann kostenlosen Support, und werden bei Updates oder Upgrades informiert.

Unsere Programme werden auf zweiseitig formatierten Disketten geliefert. Sollten Sie kein zweiseitig arbeitendes Laufwerk besitzen, überspielen wir Ihnen Ihr Programm gerne auf einseitig formatierte Disks. Dazu senden Sie uns Ihre Diskette(n) bitte mit einem adressierten Rückumschlag zu.

2 Befindet sich auf einer Diskette eine Datei READ_ME oder LIESMICH, lesen Sie diese bitte zuerst durch.
Bei einigen Produkten werden Sie beim ersten Aufruf des Programms aufgefordert, Ihren Namen und Ihre Adresse einzugeben. Diese Angaben werden auf Diskette gespeichert und von uns bei eventuellen Updates oder Upgrades verwendet.

3 Bevor Sie mit dem Programm arbeiten, sollten Sie dieses Handbuch unbedingt lesen. Erst dann werden Sie die Fähigkeiten des Programms voll ausschöpfen können.

Sollten Sie nach Studium des Handbuchs trotzdem Schwierigkeiten haben, wenden Sie sich an unseren technischen Support: ~~täglich~~ Donnerstag von 16.00 bis 17.00 Uhr unter der Rufnummer 07231/35 60 35 oder schriftlich, aber nur mit beigefügtem Support-Formular. Um Ihre Anfragen beantworten zu können, benötigen wir unbedingt folgende Informationen:

- Systemkonfiguration (Rechner, Harddisk, Speicher, Zusatzhardware etc.)
- Monitor, Auflösung
- Typ und Version von Betriebsystem und Harddisk-Treiber
- Programme im AUTO-Ordner, laufende Accesories
- Version und Seriennummer des Programms
- Fehlernummer, -beschreibung
- und wie es dazu kam

Bitte haben Sie Verständnis dafür, daß wir nur registrierten Kunden diesen Service anbieten können.

Sollten Sie Ihr Programm gegen die neuste Version umtauschen wollen (Update), senden Sie uns die Originaldiskette(n) zusammen mit einem adressierten und frankierten Rückumschlag zu.
Als ausländischer Kunde müssen Sie dieser Sendung eine grüne Zollerklärung (erhältlich bei Ihrem Postamt) beifügen.

Wichtige Einleitung!

Es ist soweit: Vor Ihnen liegt das neue OMIKRON.BASIC und wartet auf seine Inbetriebnahme. Dieses Handbuch soll Ihnen helfen, Ihren Atari ST und OMIKRON.BASIC optimal zu nutzen. Wir haben das Handbuch mit viel Mühe geschrieben und hoffen, daß es verständlich ist.

Dennoch: Ein Handbuch ist keine Bettlektüre! Zumindest den ersten Teil (Erklärungen) sollten Sie *ganz* lesen - und die "Befehle, Funktionen und Operatoren" zumindest überfliegen; die Feinheiten der Beschreibungen kommen dann nach und nach, wenn Sie sie (ge)brauchen.

Wichtig: Dieses Handbuch ist kein BASIC-Lehrbuch. Grundkenntnisse vom Computern im allgemeinen und von der Programmierung in BASIC im speziellen werden vorausgesetzt.

In den OMIKRON.Basic-Interpreter 3.5 ist ein "kontextsensitives" Hilfe-System eingebaut. Sie können auf verschiedene Arten Hilfe zu den einzelnen Befehlen und Funktionen bekommen. Steht ein Cursor auf einem Befehl oder einer Funktion und Sie drücken die HELP-Taste, so bekommen Sie entsprechende Erläuterungen. Diese Erläuterungen können mit dem mitgelieferten MAKEHELP.PRG geändert und mit eigenen Notizen ergänzt werden.

Drücken Sie die HELP-Taste, ohne daß dem Programm deutlich ist, *wozu* Sie gerade Hilfe suchen, werden Sie entsprechend gefragt. Es lohnt sich also, ab und zu ganz neugierig auf HELP zu drücken ....

Telefonische Hilfe bekommen Sie während unserer Suport-Zeiten (derzeit, also im Januar 1992: Mo, Mi, Fr jeweils von 14-17 Uhr unter 07231-356035) - am besten, *nachdem* Sie das Handbuch gelesen haben ...

Auf Ihrer Programmdiskette finden Sie einen Ordner mit Demo-Programmen, was Ihnen einerseits viel eigenes Herumprobieren ersparen mag - und Sie anderseits dazu ermuntern soll.

Vor dem ersten 'Herumprobieren' sollten Sie eine evtl. auf der Diskette befindliche README- oder LIESMICH-Datei lesen.

Bitte kopieren Sie OM-BASIC.PRG nicht in einen AUTO-Ordner! Programme, die im AUTO-Ordner stehen, werden nicht unter GEM gestartet. Sie hätten im Omikron.BASIC.PRG dann keine Alert-Boxen für Lesefehler auf Disk; alle Grafikbefehle funktionierten nicht und einiges andere ebensowenig ...

OM-BASIC.PRG wird ganz normal per Doppelklick gestartet.

Gutes Lernen - und viel Erfolg mit Ihren eigenen Basic-Programmen!

Inhaltsverzeichnis

Teil 1: Erklärungen:

Der Full-Screen-Editor

Die Menüpunkte im Einzelnen

Erklärungen zum Handbuch

Gliederung des Handbuches

Das Handbuch ist in drei Teile gegliedert:

1. Erklärungen
2. Referenzen
3. Anhang.

Im ersten Teil werden vor allem die beiden Editoren des OMIKRON.BASIC erläutert.

Im Referenzteil (dem größten Teil dieses Handbuchs) sind alle Befehle in alphabetischer Reihenfolge aufgeführt.

In den Anhängen finden Sie Tabellen und interne Informationen über OMIKRON.BASIC.

Dieses Handbuch soll also überwiegend als *Nachschlagewerk* dienen. Zum Lernen der Programmiersprache ist es nicht geeignet. Hierfür empfehlen wir entsprechende Fachbücher (z.B. "Das große OMIKRON.BASIC-Buch", Heim-Verlag, Darmstadt).

Werden an einigen Stellen besonders "technisch-tiefschürfende" Hinweise gegeben, so sind sie mit "Für Profis:" gekennzeichnet. Im allgemeinen werden Sie auch ohne das Verständnis dieser Hinweise auskommen.

Tasten

Zu drückende Tasten stehen fast immer in eckigen Klammern:
[Return], [K], [Help], [CTRL] für Control usw.

Die Pfeil-Tasten auf dem Cursor-Block sehen im Handbuch so aus:
[↑] [↓] [→] [←]

Manche Tasten kommen mehrfach auf der Tastatur vor - sowohl auf dem DIN-Block (links) als auch auf dem Ziffern-Block (rechts).

Gewöhnlich ist es gleich, ob Sie die Taste auf dem DIN- oder dem Ziffern-Block drücken; wenn nicht, steht ausdrücklich dabei, welcher Block

gemeint ist. Eine Besonderheit ist die Taste [Return]; sie heißt auf dem Ziffernblock [Enter], hat aber genau die gleiche Funktion.

Es gibt einige Tasten, die (fast) nur in Kombination mit anderen Tasten gedrückt werden:

[Shift]

[Alternate], abgekürzt [ALT]

[Control], abgekürzt [CTRL].

Wenn Sie mehrere Tasten gleichzeitig drücken sollen, steht zwischen den beiden Tasten ein Bindestrich:

[CTRL]-[Insert] oder [Alternate]-[@]

Schriftstile

In diesem Handbuch verwenden wir neben der normalen Schrift noch diese Schrift für:

- Programme zum Eintippen
- BASIC-Befehle
- Ausgaben des Computers am Bildschirm.

Zur Groß/Kleinschreibung

Befehle sind hier im Handbuch immer komplett im Großbuchstaben geschrieben:

PRINT, OPEN, PCIRCLE etc.

Bei Variablen, Labels, selbstdefinierten Funktionen und Prozeduren etc. ist hingegen nur der erste Buchstabe groß, der Rest sind Kleinbuchstaben: (Ausnahme: Nach einem Unterstrich "_" wird der darauffolgende Buchstabe wieder großgeschrieben).

Variablen: Anzahl, Masse etc.

Prozedur bzw. Funktion: z.B. Type("Name"), FN Binomial(X,Y)

Diese Form der Ausgabe wird auch im Listing verwendet.

Beim Programmieren spielt es keine Rolle, welche Form der Groß- und Kleinschreibung Sie verwenden. Die oben Angegebene ist lediglich die Form, die vom LIST-Befehl verwendet wird.

Syntax-Beschreibungen

Syntax-Beschreibungen enthalten einige Sonderzeichen, die einer Erklärung bedürfen.

<Zahl> Worte in diesen Klammern sind Beispiele. In diesem speziellen Fall könnte man z.B. 3 einfügen.

[STEP] Worte in diesen Klammern können ausgelassen werden.

{;|,} Die Teilstücke in diesen Klammern sind Auswahlen. Nur eines der Teilstücke darf hier benutzt werden.

A[A,...] Eine Syntax dieser Form bedeutet, daß an dieser Stelle beliebig viele A's eingegeben werden dürfen.

TEXT In diesem Fall sollten Sie nicht einfach irgendeinen Text eintippen. Hier geht es darum, einfach ganz genau [T][E][X][T] einzutippen.

<Parameterliste> Oft wird eine Syntax wie <Parameter>,[<Parameter>...] so abgekürzt: <... Liste>

Kompatibilität zu MBASIC

Wenn Sie bereits Microsofts MBASIC kennen, dann wird Ihnen der Umgang mit OMIKRON.BASIC nicht schwer fallen: OMIKRON.BASIC ist zu 99% MBASIC-kompatibel. Das bezieht sich auf die Übertragung von MBASIC-Programmen auf OMIKRON.BASIC und nicht etwa umgekehrt: Unter OMIKRON.BASIC haben Sie einige Befehle mehr zur Verfügung. Obwohl wir uns Mühe gegeben haben, kompatibel zu MBASIC zu bleiben, gibt es doch einige Dinge, die MBASIC von OMIKRON.BASIC unterscheiden. Diese Unterschiede stellen wir Ihnen hier in Kürze vor.

LOG - in MBASIC berechnet LOG(x) den natürlichen Logarithmus. Unter OMIKRON.BASIC heißt dieser Befehl LN(x).

MKS$, MKD$ - diese Funktionen wandeln Zahlen in das interne Fließkomma-Format um. Weil die Fließkommaformate der beiden BASIC-Interpreter nicht gleich sind, sind auch die Ergebnisse der beiden Funktionen in MBASIC und OMIKRON.BASIC nicht gleich.

CVS,CVD - aus den oben angeführten Gründen sind natürlich auch die Umkehrfunktionen zu MKS$ und MKD$ verschieden.

Diese Inkompatibilitäten fallen aber nur auf, wenn in den mit MKS$ oder MKD$ erzeugten Strings "von Hand" manipuliert wird.

DATA - wenn Sie Stringdaten ohne Anführungszeichen in MBASIC in DATA-Zeilen einsetzen, werden diese als Strings behandelt. Unter

OMIKRON.BASIC wird ein solcher Stringausdruck ohne Anführungsstriche (z.B.: HAUS) als Variable gelesen. Ein READ A gibt dann den Inhalt der Variablen HAUS zurück, ein READ A$ führt zu einem TYPE MISMATCH ERROR.

DEFINT,DEFSNG,DEFDBL,DEFSTR müssen in OMIKRON.BASIC ganz am Anfang eines Programmes stehen und auch als erste Befehle eingetippt werden!

Alle normalen Variablen (A,B,Haus,Laufvariable etc.) sind in OMIKRON.BASIC automatisch long-integer!!! Wenn Sie mit Float-Variablen arbeiten wollen, müssen Sie DEFSNG "A-Z" als erste Zeile ins Programm einfügen oder hinter jede Variable das Postfix ! stellen (A!,B!,Haus!,Laufvariable!).

RANDOMIZE gibt es in OMIKRON.BASIC nicht. Lassen Sie diesen Befehl in Programmen, die Sie übertragen wollen, einfach weg.

RND(-X) gibt nicht den Startwert des Zufallsgenerators an, sondern eine ganzzahlige Zufallszahl.

WIDTH gibt es in OMIKRON.BASIC nicht.

ERASE gibt es in OMIKRON.BASIC nicht. Sie können ein Feld mit einem DIM-Befehl redimensionieren.

CHAIN ...,ALL gibt es in OMIKRON.BASIC nicht.

Der Formatstring bei PRINT USING ist in OMIKRON.BASIC etwas anders aufgebaut. Genaueres lesen Sie bitte unter PRINT USING nach.

CLOSE löscht Strings in Dateibuffern, wie beim MBASIC-Compiler.

INPUT kann nur ganze Zeilen einlesen. Drei durch Komma getrennte Zahlen können nicht mit

```
INPUT A:INPUT B:INPUT C
```

eingelesen werden, sondern nur mit

```
INPUT A,B,C
```

Der Full-Screen-Editor

Nach dem Sie OMIKRON.BASIC gestartet haben, haben Sie den sogenannten Full-Screen-Editor vor sich: Am oberen Bildschirmrand sehen Sie eine Menüzeile, ansonsten ist der Bildschirm komplett leer, bis auf eine Dialogbox in der Mitte des Bildschirms mit der Versionsnummer von OMIKRON.BASIC. Diese Box verschwindet, sobald Sie eine Taste (auch Maustaste) drücken oder ein Menü auswählen.

Im Full-Screen-Editor können Sie Ihren Programmtext eingeben oder bearbeiten. Der Full-Screen-Editor betrachtet alle Ihre Eingaben - egal, ob Sie Zeilennummern eingeben oder nicht - als Programm.

Direkte Anweisungen werden also wie Programmzeilen ohne Zeilennummer behandelt. Weil es in diesem Editor zwei verschiedene Eingabemodi gibt, ist die Anweisung entweder ein Fehler (weil ihr keine Zeilennummer voransteht) oder eine vollkommen korrekte Programmzeile (weil Sie die Zeilennummern abgeschaltet haben).

Abgesehen vom bloßen Eintippen von Text, was jeder Editor beherrscht, bietet dieser Editor eine ganze Serie von Spezialfunktionen, die dazu gedacht sind, das Eintippen und Ändern eines Programms zu vereinfachen. Dies beinhaltet automatische Suchfunktionen (wo habe ich die Variable A schon einmal verwendet?), Blockfunktionen und eine eingebaute Hilfestellung zu allen Befehlen/Funktionen und dem Editor selbst.

Natürlich gibt es auch Funktionen, um ein fertiges Programm abzuspeichern, es zu starten oder auch den Editor zu verlassen. Die meisten Funktionen können sowohl mit der Maus als auch mit der Tastatur eingegeben werden.

Wichtige Tasten im Full-Screen-Editor

Das Hilfe-System

[Help] aktiviert die eingebaute Hilfefunktion. Hierbei gibt es mehrere Möglichkeiten:

- Sie drücken [Help]. Es erscheint eine Dialogbox am Bildschirm, in der sie einen Hilfetext lesen können. In diesem Text sind einzelne Wörter **fett** dargestellt. Durch Mausklick auf eines dieser Wörter wird ein dazugehöriger Hilfetext aufgerufen.
- Sie drücken [Help] und der Cursor steht auf einem BASIC-Befehl oder einer BASIC-Funktion. In der erscheinenden Dialogbox am Bildschirm steht die Erklärung zu genau diesem Befehl.
- Sie drücken [Help] und haben gerade ein Menü angewählt, aber noch nicht die Maustaste gedrückt. Die Dialogbox erklärt diesen Menüpunkt.

Nähere Informationen über die Bedienung dieser Dialogbox finden Sie später.

Einfüge- und Überschreib-Modus

Grundsätzlich werden alle Zeichen, die Sie eintippen an der Stelle am Bildschirm dargestellt, an der der Cursor steht. Der Editor arbeitet dabei im ***Insert-Modus*** (engl.: insert - einfügen) oder im ***Overwrite-Modus*** (engl.: overwrite - überschreiben).

Zwischen Insert- und Overwrite-Modus können Sie leicht unterscheiden: Im Insert-Modus sieht der Cursor wie ein senkrechter Strich aus. Im Overwrite-Modus ist er ein blinkendes Rechteck. Vom Overwrite- in den Insert-Modus (und zurück) kommen Sie über die Tastenkombination [Control]-[Insert].

Die Umkehrung zu [Delete] ist [Insert]: Es wird ein Leerzeichen eingefügt, und der Rest der Zeile hinter der Cursorposition rutscht um ein Zeichen nach rechts.

Einfügen von Zeilen

Im Insert-Modus können ganze Zeilen einfach durch Betätigen der [Return]-Taste eingefügt werden. Die Leerzeile erscheint immer unterhalb der aktuellen Zeile. Der Cursor wird an den Anfang dieser Zeile gestellt. Wenn Sie also einige Zeilen in Ihr Programm einfügen wollen, können Sie einfach mehrmals [Return] drücken.

Die Tastenkombination **[Control]-[Return]** bewirkt eine etwas andere Funktion: Hiermit wird die aktuelle Zeile in zwei Zeilen aufgespalten.

Wenn Sie am Ende einer Zeile [Delete] drücken, wird die folgende Zeile an die aktuelle angehängt.

Der Cursor

Das kleine blinkende Quadrat, welches immer Ihre aktuelle Eingabeposition markiert, nennt sich Cursor. Mit den Pfeiltasten können Sie die Position des Cursors verändern, durch Mausklick auf die gewünschte Position ebenso.

Mit **[Tab]** rücken Sie den Cursor auf die nächste 8er-Spalte vor.

Mit **[Clr Home]** bewegen Sie den Cursor an die erste Zeile des Programmtextes. Ist der Cursor schon in der ersten Zeile, so springt er durch [Clr Home] zur letzten Zeile. Sind Sie mitten im Programmtext, so können Sie durch zweimaliges [Clr Home] also an den Schluß des Programmes kommen.

An den Start Ihres Programms können Sie auch kommen, indem Sie **[Control]-[Shift]-[↑]** drücken. Analog kommen Sie zur letzten Zeile mit [Control]-[Shift]-[↓].

[Control]-[→] und **[Control]-[←]** bewegen den Cursor wortweise im Text. [Control]-[Shift]-[→] und [Control]-[Shift]-[←] bewegen den Cursor an das Ende oder den Anfang einer Textzeile.

Mit **[Control]-[1] bis [Control]-[4]** setzen Sie eine Marke an der aktuellen Cursorposition. Diese Marken können Sie mit **[Alternate]-[1] bis [Alternate]-[4]** wieder anspringen. Bitte beachten Sie: Die Ziffern "1" bis "4" müssen mit den Zifferntasten auf der Haupttastaur eingegeben werden.

[Control]-[Delete] löscht den Rest der Zeile ab der Cursorposition.

[Control]-[Shift]-[Delete] löscht die ganze aktuelle Zeile, ohne daß die nächste Zeile hochrückt.

[Shift]-[F9] oder **[CTRL]-[Y]** machen fast das gleiche; hier rutschen die Zeilen unter dem Cursor aber nach oben.

Sonstige Funktionen

[Alternate] alleine, ohne eine andere Taste gedrückt, hat zwei Funktionen:

- Abbrechen der Wiederholfunktion (Taste [F6]).
- Abbrechen von Such- oder Ersetzfunktionen ([F2] oder [F3]).

Wenn Sie vor dem Verlassen einer Zeile **[Undo]** drücken, werden die Änderungen an dieser Zeile rückgängig gemacht. Außerdem bricht [Undo] die meisten Funktionen ab, wenn Sie aus Versehen angewählt wurden.

Wenn Sie Texte eintippen, könnte es sein, daß Sie einmal Zeichen eintippen wollen, die normalerweise als Steuerzeichen dienen. Für solche Fälle geben Sie **[CTRL]-[A]** **<[Taste]>** ein. Dies ist z.B. bei der **[Esc]-Taste** nützlich, wenn Sie Druckersteuercodes in Ihrem Programm benötigen.

Die Dialogboxen

Wann immer der Full-Screen-Editor Eingaben von Ihnen benötigt oder Ihnen etwas mitteilen will, kommen Dialogboxen zum Einsatz. Alle diese Dialogboxen sind auch per Tastatur bedienbar. Hierzu befindet sich in jedem Knopf in der linken oberen Ecke ein kleines Zeichen. Wenn Sie die [Alternate]-Taste zusammen mit der entsprechenden Taste drücken, wird die Funktion genauso ausgelöst wie per Mausklick.

Eine Taste ist in jeder Dialogbox möglich: Mit **[Undo]** beenden Sie die Dialogbox, ohne daß sich etwas ändert. Sie können also eine versehentlich aufgerufene Dialogbox immer mit [Undo] 'loswerden'.

Bitte beachten Sie: Wenn in einem Knopf eine kleine Ziffer steht, ist auch hier die Haupttastatur gemeint.

Alle anderen Funktionen verhalten sich genauso wie bei herkömmlichen GEM-Dialogboxen.

Eine Besonderheit stellt die Dialogbox des Hilfesystems dar:

Fettgedruckte Wörter sind Verweise auf andere Teile der Hilfe. Führen Sie auf einem fettgedruckten Wort einen Mausklick aus, wird sofort die Hilfestellung für dieses Wort geladen.

Pfeiltasten am rechten Rand blättern in dem Hilfetext eine Zeile nach oben bzw. unten. Diese Funktion kann auch durch einen Druck auf die Pfeiltasten der Tastatur erreicht werden.

"Previous": Die Hilfefunktion "merkt" sich die letzten Hilfestellungen. Durch einen Klick auf den Knopf "Previous" können Sie sich die letzten Hilfestellungen nochmals anzeigen lassen.

"Exit":Über den Knopf "EXIT" (oder die Taste [Undo]) verlassen Sie das Hilfesystem.

Die Dateiauswahlbox

Zur Dateiauswahl wird die gewöhliche Box des Betriebssystems verwendet. Es gibt jedoch eine Besonderheit: wird die Box ohne Auswahl eines Namens verlassen, wird der aktuelle Pfad auf den gerade geählten Pfad umgestellt (z.B. beim Menü "LOAD"). Entsprechend wird bei "LOAD BLOCK" der Block-Pfad und die Block-Extension vermerkt, wenn Sie nach Einstellen des Pfades die Box ohne Namen mit "OK" verlassen. Sie können somit also dafür sorgen, daß Blöcke z.B. immer die Extension "*.BLK" tragen und zunächst im Verzeichnis "P:\" gesucht werden. Das schafft Ordnung und verhindert Verwechslungen zwischen Token-Format ("*.BAS") und ASCII-Format bei Blöcken ("*.BLK").

Die Maus im Full-Screen-Editor

Die Maus dient zum komfortablen Markieren und Kopieren vom Blöcken. Man kann die Cursorposition neu setzen und schnell nach Querverweisen suchen.

Markieren von Blöcken: Drücken Sie die linke Maustaste. Bewegen Sie nun bei gedrückter Maustaste den Mauszeiger über den Text. Der überstrichene Text wird sofort schwarz unterlegt und somit als Block markiert.

Man unterscheidet zwei Arten von Blöcken: Den *Spaltenblock und den Zeilenblock.*

Wenn Sie den Mauszeiger beim Markieren nur innerhalb einer Zeile bewegen, wird nur ein Teil dieser Zeile markiert (Spaltenblock). Verlassen Sie diese Zeile, so wird diese Zeile und alle folgenden als ganzes markiert (Zeilenblock). Um nur eine Zeile zu markieren, muß der Mauszeiger ganz am linken Rand geführt werden.

Einfügen von Blöcken: Sobald ein Block markiert ist, wird er beim Drücken der rechten Maustaste an der Cursorposition eingefügt.

Positionieren der Maus: Ein kurzer Klick mit der linken Maustaste stellt der Cursor auf die neue Position.

Suchen von Querverweisen: Ein Doppelklick mit der linken Maustaste auf ein Wort (Basic-Befehl, Prozedurname oder Variable) löst eine Suche nach eben diesem Wort aus. Es wird immer nach einem Token, also nach einem Basic-relevanten Wort, gesucht. Begriffe in Anführungszeichen werden nicht gefunden (siehe auch Menü "FIND TOKEN"). Es erscheint eine Liste der gefundenen Textstellen an die durch Anklicken verzweigt werden kann. Wenn die neue Textstelle nun angezeigt wird kehrt man auf Wunsch mit [Control]-[Z] zum

Ausgangspunkt zurück. Auf diese Weise kann man sich sehr schnell einen Überblick im Text verschaffen (z.B. "Wo ist eine Prozedur definiert, wo wird sie verwendet?"). Das Zurückkehren mit [Control]-[Z] funktioniert mehrfach. Es werden bis zu acht Textstellen vermerkt. Man kann also mehrfach durch Doppelklick suchen und weiter verzweigen und sich durch mehrfaches Drücken von [Control]-[Z] bis zum Ausgangspunkt "zurückhangeln".

Die Menüpunkte im Einzelnen

Die Menü-Zeile

Die oberste Zeile ist wie bereits erwähnt die Menü-Zeile des Editors mit den Überschriften FILE, FIND, BLOCK, MODE, GO und RUN. Dieses Menü können Sie genauso bedienen wie ein GEM-Menü.

Im Gegensatz zu normalen GEM-Menüs können Sie hier auch *auf die Überschriften klicken.* Es wird hierbei die jeweils erste Funktion im entsprechenden Menü ausgelöst. Im FIND-Menü wäre dies z.B. die Funktion FIND.

Viele Menüpunkte können auch über die Tastatur ausgelöst werden (z.B. [Control]-[O] für "Load..."). Wenn ein Menüpunkt auch über eine Tastenkombination ausgelöst werden kann, steht diese Kombination rechts neben dem Menüeintrag. Hierbei gelten folgenden Abkürzungen: "^" steht für [Control], "◆" steht für [Alternate], "[↑]" steht für [Shift].

Das FILE-Menü:

"NEW" [Control]-[N]

Löscht das im Speicher befindliche Programm. Falls das Programm seit dem letzten Speichern geändert wurde, erfolgt eine Sicherheitsabfrage. Die Knöpfe der erscheinenden Dialogbox haben hierbei folgende Bedeutung:

"Yes" : Das Programm wird vor dem Löschen gespeichert.

"No" : Das Programm wird gelöscht.

"Cancel": Die Funktion "NEW" wird abgebrochen.

Ob sich das Programm geändert hat, können Sie übrigens auch selbst jederzeit feststellen: Im Falle einer Änderung gibt der Editor vor dem Dateinamen einen Stern "*" aus. Sobald das Programm gesichert ist, verschwindet der Stern wieder.

"LOAD" [Control]-[O]

Lädt ein Programm von Diskette/Festplatte. Das Programm kann im internen Tokencode oder als ASCII-Datei vorliegen. Es erscheint eine Dateiauswahlbox, in der Sie das gewünschte Programm auswählen können.

"LOAD BLOCK" [F7]-[Shift]-[F8]

Lädt einen Block von Diskette/Festplatte und fügt ihn an der aktuellen Cursorposition ein. Der Block muß im ASCII-Format vorliegen und sollte am besten mittels "SAVE BLOCK" erstellt worden sein. Es erscheint eine Dateiauswahlbox, in der der Name des Blockes ausgewählt werden kann.

"SAVE" [Control]-[S]

Speichert das im Speicher befindliche BASIC-Programm unter dem aktuellen Namen im internen Tokencode ab. Außerdem werden die Marken im Programm mit abgespeichert und sind nach dem Laden sofort wieder verfügbar (für die Dauer-Verwendbarkeit vom Marken wichtig). Desweiteren werden mit dem Programm abgespeichert: Die Cursorposition, die Blockmarken und die Bildschirmeinstellungen (Font/Grösse ...). All dieses ist nach dem Laden genau wieder wie beim Abspeichern des Programms (erheblich komfortabler als bei älteren BASIC-Versionen).

Der aktuelle Name (unter dem das Programm gespeichert wird,) wird in der rechten oberen Ecke, neben den Menütiteln angezeigt. Lautet dieser Name "NONAME.BAS" erscheint beim Speichern eine Dateiauswahlbox. In allen anderen Fällen wird das Programm ohne Dateiauswahlbox abgespeichert.

"SAVE AS" [Control]-[M]

Speichert das im Speicher befindliche Programm im internen Tokencode ab. Im Gegensatz zu "SAVE" wird bei dieser Funktion *jedesmal* mittels einer Dateiauswahlbox der Name der Datei erfragt. Dieser Name wird dann zu dem aktuellen Namen erklärt und rechts oben angezeigt. Ansonsten gilt das unter "SAVE" Gesagte.

"SAVE BLOCK" [F7]-[F8]

Speichert den zur Zeit markierten Block (s. BLOCK-Menü) im ASCII-Format ab. Hierzu wird der Dateiname über eine Dateiauswahlbox erfragt. Ein so gespeicherter Block kann mittels "LOAD BLOCK" wieder eingelesen werden.

"PRINT" [Control]-[P]

Druckt das im Speicher befindliche Programm auf einem angeschlossenen Drucker aus. Es erscheint eine Dialogbox, in der die Breite des Ausdrucks (Anzahl Zeichen pro Zeile) eingegeben werden kann. Anschließend wird das Programm ausgedruckt. Falls eine Zeile des Programms länger als die angegebene Maximallänge ist, wird sie abgeschnitten und in der nächsten Zeile fortgesetzt. Allerdings wird sie in der nächsten Zeile soweit eingerückt, daß sie unter dem Anfang der letzten Zeile - also hinter der Zeilennummer - steht.

"DIRECTORY"

Gibt das Inhaltsverzeichnis eines Laufwerks/Ordners aus. Es erscheint eine Dialogbox, in der angegeben werden kann, welcher Pfad und welche Dateien angezeigt werden sollen. Es gelten hier die gleichen Regeln wie bei der Dateiauswahlbox. Sollen z.B. alle Programme im Ordner "COMPILER" auf dem Laufwerk D: angezeigt werden, müssen Sie eingeben:

D:\COMPILER*.PRG

Anschließend wird das Inhaltsverzeichnis angezeigt. Nach jeweils einer Bildschirmseite werden Sie aufgefordert, eine Taste zu drücken. Nachdem alle Dateien angezeigt worden sind, wird nochmals auf einen Tastendruck gewartet und anschließend in den Editor zurückgesprungen.

"DIRECT MODE" [Control]-[C]

Springt in den Direktmodus - einen Bildschirmeditor, der weiter unten näher beschrieben wird. In diesem Bildschirm werden übrigens alle Ausgaben Ihres Programms gemacht. Wenn Sie in den Direktmodus wechseln, baut OMIKRON.BASIC den Text wieder auf, der beim letzten Mal dort stand.

Mit der Taste [Help] kommen Sie jederzeit wieder zurück in den Full-Screen-Editor.

"QUIT" [Control]-[Q]

Verläßt den Full-Screen-Editor und OMIKRON.BASIC. Falls an Ihrem Programm seit dem letzten Speichern Änderungen vorgenommen wurden, erscheint wie bei "NEW" eine Dialogbox:

"Yes" : Das Programm wird vor dem Verlassen des BASIC gespeichert.

"No" : OMIKRON.BASIC wird verlassen, das Programm nicht gesichert.

"Cancel": Die Funktion "QUIT" wird abgebrochen.

Das FIND-Menü:

"FIND" [Control]-[F]

Sucht nach einem Text innerhalb Ihres BASIC-Programms. Es erscheint eine Dialogbox, in der Sie den Suchtext eingeben können.

Es wird ab der Cursorposition gesucht. Gibt man auf die Frage SEARCH FOR: nichts ein (also nur [Return]), so wird derselbe Text wie bei der letzten Suche wieder gesucht.

Beispiel: Sie suchen die Stelle in einem Basicprogramm, an der die Fakultäts-Funktion verwendet wird. Dann drücken Sie [Clr-Home] (um nach ganz oben zu kommen) und wählen "FIND" im Menü "FIND" an; auf die Frage: SEARCH FOR: geben Sie [F][A][C][T] und [Return] ein. Der Editor findet nun (z.B): `110 PRINT FACT(N!)`

Sie suchen weiter: "FIND" anwählen und [Return] - und der Editor findet die nächste Stelle, an der die FACT-Funktion verwendet wird.

Bei der Eingabe des Suchtextes sollten Sie beachten, daß Leerzeichen vor dem Cursor anerkannt werden, hinter dem Cursor nicht.

"FIND NEXT" [Control]-[G]

Sucht ab der Cursorposition das nächste Vorkommen des Textes, den Sie bei "FIND" eingegeben haben. In obigem Beispiel könnten Sie also nach dem ersten Auftauchen von FACT auch durch Auswählen dieses Menüpunktes weitersuchen.

"FIND TOKEN" [Control]-[T]

Um diese Funktion zu erklären, muß zunächst etwas weiter ausgeholt werden: Alle bisher besprochenen Suchfunktionen können nicht zwischen Groß- und Kleinschreibung unterscheiden. Wenn Sie also nach einem kleinen "a" suchen, finden Sie auch alle großen "A"s.

Alle bisher besprochenen Funktionen suchen nach Zeichenketten. Das bedeutet, daß Sie bei der Suche nach einem "A" z.B. auch alle "DATA"-Befehle finden würden. Dies kann sehr ärgerlich sein, wenn Sie zum Beispiel überall die Variable "A" zu "Mehrwertsteuer" umbenennen wollen. Stellen Sie sich einmal vor, wie der "DATA"-Befehl danach aussehen würde!

Aus diesem Grund gibt es im FIND-Menü die vier Einträge FIND TOKEN..., LIST TOKEN..., FIND DEF... und RENAME TOKEN. Unter einem "TOKEN" versteht man im Computerdeutsch einen Bestandteil Ihres Programms (und zwar aus der Sichtweise des Computers). Hier ist also nicht ein Stück Programmtext gemeint, sondern ein BASIC-Schlüsselwort aus Ihrem Programm. Wie Sie vielleicht schon wissen, übersetzt das OMIKRON.BASIC

schon beim Eintippen ein Programm in eine kürzere, dem Computer einfacher verständliche Form. Dies betrifft nicht nur BASIC-Befehle, sondern auch Variablen, selbstdefinierte Prozeduren und Funktionen oder Labels. Jedes dieser Teile bekommt intern eine Nummer. Mit den TOKEN-Funktionen suchen Sie nach diesen internen Nummern.

Der "DATA"-Befehl hat z.B. die Nummer 27, eine Variable mit Namen "Data" hingegen hat mit Sicherheit eine andere Nummer. Natürlich finden Sie mit dieser Art der Suche nicht Bestandteile von Zeichenketten oder Ähnliches. Nur vollständige Schlüsselwörter Ihres Programms werden gefunden.

Wenn Sie also mit FIND TOKEN nach der Variable "A" suchen, finden Sie auch tatsächlich nur jedes Vorkommen dieser Variablen und nicht noch alle "DATA"-Befehle oder alle Stellen, wo z.B. "A" in einer "PRINT"-Anweisung enthalten ist. Sie finden auch nicht die Variable "Angebot$", denn diese hat ja eine andere Nummer als "A".

Hier ist eine Liste der Dinge, die Sie mit FIND TOKEN finden können:

A	- Variable
Hallo%L	- Variable
PRINT	- Befehl
Hallo()	- Feldvariable mit einem Index. Z.B. Hallo(8), aber nicht Hallo(8,3)
Hallo(,,)	- Feldvariable mit drei Indices.
-Eingabe	- Label
FN Fibonacci()	- Funktion mit einem Parameter
PROC Ausgabe	- Eine Prozedur
P Ausgabe	- Ebenfalls eine Prozedur
30	- Zeilennummer, auf die mit GOTO, GOSUB oder RESTORE verwiesen wird.

Da in einem Programm Variablen und Prozeduren die gleichen Namen haben können, ist es wichtig, bei den TOKEN-Funktionen Prozeduren ein "P" voranzustellen und Labels den Strich "-". Wenn es mehrere Prozeduren "Ausgabe" gibt, die sich nur in der Anzahl Parameter unterscheiden, dann können Sie auch dies klarmachen: "P Ausgabe()" meint die Prozedur mit einem Parameter, während "P Ausgabe(,,)" die Prozedur mit drei Parametern meint. Wenn es nur eine Prozedureoder Funktion "Ausgabe" gibt, können Sie die Parameterangaben weglassen.

FIND TOKEN verhält sich ansonsten genauso wie FIND.

Zusätzlich zu dem Menüpunkt "FIND TOKEN" gibt es noch eine weitere Möglichkeit ein bestimmtes Token zu suchen:

Der **Doppelklick auf ein Wort.** Er bewirkt im Prinzip das gleiche wie LIST TOKEN mit diesem Wort. OMIKRON.BASIC versucht allerdings selbständig herauszufinden, ob dieses Wort eine Variable, ein Label, eine Prozedur oder eine Funktion ist. Wenn das Wort mehrere Bedeutungen hat, kann es sein, daß Sie nicht das finden, was Sie gesucht haben. Versuchen Sie also bitte, eindeutige Bezeichner zu verwenden. Wenn das angeklickte Wort überhaupt nicht als Token vorkommt, wird nur die LIST TOKEN Funktion angewählt, aber noch nicht ausgeführt. Sie können dann das Wort noch editieren. (Vgl. "TO LAST MARK" im GO-Menü).

"FIND DEF"

Sucht nach der Definition eines Tokens. Wenn Sie also z.B. wissen wollen, wo die Prozedur Ausgabe_Test definiert wurde, wählen Sie "FIND DEF" an und geben in der erscheinenden Dialogbox "p Ausgabe_Test" ein. Bitte beachten Sie: Mit dieser Funktion können Sie die Definition von Prozeduren, Funktionen und die Position von Labels finden. Bitte beachten Sie: Wenn Sie eine Prozedurdefinition suchen, müssen Sie vor den Namen der Prozedur "P" schreiben, suchen Sie eine Funktionsdefiniton, schreiben Sie bitte "FN" und bei einem Label "-" vor den eigentlichen Namen.

"FIND ERROR"

Stellt den Cursor auf den nächsten SYNTAX ERROR oder TYPE MISMATCH ERROR. Logische Fehler können vom Interpreter nicht automatisch erkannt werden.

"LIST" [F2]-[F3]

Listet alle Stellen auf, an denen ein bestimmter Suchtext auftritt. Sie können damit leicht feststellen, an welchen Stellen Sie z.B. die Variable Eingabe$ verwendet haben.

Nach Aufruf dieser Funktion erscheint eine Dialogbox in der Sie den Suchtext eingeben können. Wenn Sie den "OK"-Knopf anklicken oder [Return] drücken werden die gefundenen Texte aufgelistet. Wenn Sie den Knopf "PRINTER PROTOKOLL" anklicken, werden die gefunden Texte auf dem angeschlossenen Drucker ausgegeben. Wollen Sie dies doch nicht können Sie diese Funktion durch nochmaliges Anklicken von "PRINTER PROTOKOLL" wieder abschalten.

Wenn Sie doch nicht suchen wollen, brechen Sie mit [Undo] ab.

"LIST TOKEN" [F2]-[Shift][F3]

Sucht nach BASIC Schlüsselwörtern (Tokens). Wie bei "LIST" können Sie auch hier das gewünschte Token in einer Dialogbox eingeben. Ähnlich wie bei "FIND TOKEN" sind hier auch "P", "FN" bzw. "-" für Prozeduren, Funktionen und Labels erlaubt.

"REPLACE" [F3], nochmal [F3]

Sucht einen bestimmten Text und ersetzt ihn durch einen anderen Text. In der Dialogbox geben Sie sowohl den Such- als auch den Ersatztext ein. Außerdem können Sie durch Anklicken der Knöpfe "Query" bzw. "All" den Ersetzmodus festlegen. Ein Druck auf [Return] oder den "OK"-Knopf beginnt die Ersetzung; mit [Undo] brechen Sie die Funktion wie gewöhnlich ab.

Je nach eingestelltem Modus wird nun folgendermaßen vorgegangen:

Query: Bei jedem Vorkommen des Suchtextes wird gefragt, ob dieser Text ersetzt werden soll oder nicht, oder ob die Funktion abgebrochen werden soll.

All: Es wird einfach im gesamten Text der Suchtext durch den Ersatztext ersetzt.

"RENAME TOKEN"

Benennt Tokens, also Variablen, Prozedur- und Funktionsnamen und Labels um. Natürlich lassen sich mit dieser Funktion auch Befehlstokens umwandeln, also z.B. PRINT in INPUT.

Primär ist diese Funktion aber zum Umbenennen von Variablen gedacht.

Auch Prozeduren und Funktionen lassen sich auf diese Weise umbenennen. Es erscheint eine Dialogbox, in der Sie sowohl den alten als auch den neuen Namen des umzubenennenden Tokens angeben. Geben Sie Prozedur- oder Funktionsnamen mit den jeweiligen Präfixen P oder FN an. Bei "NEW NAME" dürfen Sie keine Prä- oder Postfixe mehr verwenden!

Das BLOCK-Menü:

Sie können sich im Editor Blöcke definieren und mit diesen arbeiten, z.B. den Block löschen, kopieren, abspeichern etc. Das geschieht mit den Blockkommandos, die Sie im BLOCK-Menü finden.

"INSERT" [F7]-[F9]

Diese Funktion kopiert den Block an die aktuelle Cursorzeile. Dabei wird der Text, der schon an dieser Stelle steht, nicht überschrieben, sondern der Block wird eingefügt.

Der ursprüngliche Block bleibt unverändert.

"MOVE" [Control]-[V]

Der aktuelle Block wird an die aktuelle Cursorposition gebracht. An der alten Position wird er dabei gelöscht.

"KILL" [Control]-[E]

Löscht den aktuellen Block ***ohne Rückfrage***. Bitte beachten Sie: Es ist nicht möglich, einen gelöschten Block wieder zurückzuholen.

"BLOCK START" [Control]-[B]

Markiert die derzeitige Cursorzeile als Anfang eines Blocks.

"BLOCK END" [Control]-[K]

Markiert die derzeitige Cursorzeile als Ende eines Blocks.

Einfacher ist es, Blöcke mit der Maus markieren: Am einen Ende des Blockes Maustaste drücken, am anderen wieder loslassen.Nachdem Sie den Block definiert haben, erscheint dieser invers.

Es ist nicht möglich, einen Block zu definieren, der mitten in einer Zeile anfängt und irgendwo inmitten einer anderen Zeile aufhört.

"HIDE" [Control]-[H]

Die Markierung des aktuellen Blocks wird gelöscht.

"LOAD BLOCK" [F7]-[Shift]-[F8] und **"SAVE BLOCK" [F7]-[F8]**

Tauchen der Vollständigkeit halber sowohl im Menü als auch hier im Handbuch nochmals auf. Das andere mal siehe unter "File"- Menü.

"PRINT BLOCK" [Shift]-[Control]-[P]

Der aktuelle Block wird auf den Drucker ausgegeben. Vorher haben Sie noch die Gelegenheit, die Zeilenbreite einzustellen, die Ihrem Drucker entspricht. Voreingestellt ist 80 Zeichen. Wenn Sie Ihren Drucker nicht auf

eine engere Schrift eingestellt haben, können Sie hier einfach [Return] drücken. (siehe auch "PRINT" im "FILE"-Menü)

Das MODE-Menü:

"SWITCH SCREEN" [Shift]-[F2]

Springt zwischen den beiden mit "SPLIT SCREEN" (s.u.) eingestellten Bildschirmen hin und her.

"SPLIT SCREEN" [Shift]-[F1]

Spaltet den Bildschirm waagrecht in zwei unabhängige Teile auf.

Diese beiden Bildschirme sind völlig unabhängig voneinander. Wenn Sie also in dem einen Bildschirm den Anfang Ihres Programms darstellen, können Sie in dem anderen Bildschirm das Ende (oder eine andere Stelle) Ihres Programms ansehen. Allerdings beziehen sich beide Bildschirme auf nur ein Programm, d.h. der Editor kann trotz dieser Aufspaltung nach wie vor nur ein Programm bearbeiten.

Manchmal ist es nicht sinnvoll, daß die beiden Bildschirmteile gleich groß sind. Deshalb können Sie die Größe der Teilbereiche einfach ändern indem Sie mit der Maus den Trennstrich anklicken und bei gedrückter Maustaste in eine andere Position ziehen.

"CHANGE SIZE/FONT" [Shift]-[F3]

Schaltet auf einen anderen Zeichensatz um.

Im Editor des OMIKRON.BASIC können Sie statt der normalen Aufteilung in 25 Zeilen zu je 80 Spalten auch noch andere Bildschirmauflösungen wählen. Je nach Bildschirmauflösung und geladenen GDOS-Zeichensätzen können Sie hier verschiedene Schriftgrößen einstellen. Nur proportionale Zeichensätze werden nicht verwendet.

Über diesen Menüpunkt lassen sich lediglich die Zeichensätze umschalten. Um auf einen anderen Bildschirmtreiber umzuschalten (also VDI-Ausgabe oder OMIKRON.Ausgabe zu verwenden) benutzen Sie bitte den Menüpunkt "PREFERENCES" im "MODE"-Menü. (Für eine nähere Beschreibung s. dort.)

Wenn Sie diesen Menüpunkt anwählen werden alle vorhandenen Zeichensätze (auch geladene GDOS-Fonts, falls GDOS vorhanden ist) eingestellt.

HINWEIS: GDOS-Fonts werden nur bei aktiviertem VDI-Treiber dargestellt; außerdem muß GDOS geladen sein!

"PREFERENCES" [Control]-[L]

Legt verschieden Grundeinstellungen des Editors fest. In diesem Menüpunkt sind einige Menüpunkte aus dem MODE-Menü des OMIKRON.BASIC vor Version 3.5 zusammengefaßt.

Wenn Sie diesen Menüpunkt aufrufen, erscheint eine Dialogbox, in der Sie die folgenden Einstellungen tätigen können:

Screen Output: Hier bestimmen Sie den Bildschirmtreiber. DIRECT bezieht sich dabei auf den OMIKRON.Treiber, VDI auf den VDI-Treiber. Sie sehen hier auch, welcher Treiber gerade aktiv ist.

Normalerweise werden Sie den auf Geschwindigkeit hochoptimierten OMIKRON.Treiber (einzustellen über DIRECT) verwenden. Dieser arbeitet auf Grund seiner hohen Ausgabegeschwindigkeit nur in den Standardauflösungen von ST bzw. TT. Wenn Sie eine Grafikkarte von einem Fremdanbieter verwenden oder auf eine andere Weise die Bildschirmauflösung Ihres Rechners verändert haben, verweigert der OMIKRON.-Treiber seinen Dienst. Zu diesem Zweck wurde der VDI- Treiber eingebaut (zu aktivieren über VDI). Dieser Treiber funktioniert auf allen Grafikkarten und in allen Auflösungen. (sofern die entsprechende Grafikkarte sich an die ATARI-Richtlinien hält; sollte auch der VDI-Treiber bei Ihnen nicht funktionieren, wenden Sie sich bitte NICHT an uns, sondern an den Hersteller Ihrer Grafikkarte; der Fehler liegt eindeutig dort!) Leider ist die Ausgabegeschwindigkeit des VDI-Treibers nicht allzu hoch. Dies liegt daran, daß der Treiber für alle möglichen Fälle konzipiert wurde und daher nur die (langsamen) Standardroutinen des ST/TT für Bildschirmausgaben verwendet.

Edit Mode: Stellt den Insert- bzw. Overwrite-Modus ein. Auch hier können Sie ablesen, welcher Modus gerade aktiv ist.

Linenumbers: Schaltet die Zeilennummern ein (ON) oder aus (OFF). Wenn die Zeilennummern eingeschaltet sind, müssen Sie vor jeder Zeile eine Zeilennummer schreiben, wenn Sie nicht die Fehlermeldung "Bad Linenumber" erhalten wollen. Zusätzlich können Sie hier den aktuellen Modus nachsehen. Wenn Sie die Zeilennummern abschalten (Klick auf OFF) gelten folgende Regeln:

- Bei der Eingabe einer Zeile ist keine Zeilennummer mehr nötig.
- SAVE und BLOCK SAVE speichern ohne Zeilennummern ab.

- LOAD und LOAD BLOCK laden auch Programme ohne Zeilennummern ein.
- Bei [F1] (GO) und [F2] [F3] (LIST) werden die Zeilen wie in Einerschritten durchnumeriert betrachtet.
- Beim Verlassen des Editors werden die Zeilennummern in Einerschritten durchnumeriert.
- Errorlines: Schaltet die Anzeige von fehlerhaften Zeilen ein (SHOW) bzw. aus (HIDE). Fehlerzeilen blinken bei OMIKRON.BASIC. Wenn Sie dies als störend empfinden, so können Sie dieses Blinken über diesen Dialog abstellen, indem Sie auf HIDE klicken; wieder einschalten können Sie das Blinken durch einen Klick auf SHOW.

Automatic Backup: Wenn diese Funktion aktiv (YES) ist, wird bei jedem Speichern eines BASIC-Programms eine Sicherheitskopie der alten Version mit der Endung .BAK angelegt.

Compiler: Hier können Sie den kompletten Pfad und Dateinamen des BASIC-Compilers eintragen. Wenn Sie Ihren Compiler also z.B. auf Laufwerk E: im Ordner COMPILER unter dem Namen OM_COMP.PRG liegen haben, müssen Sie hier eintragen:

E:\COMPILER\OM_COMP.PRG

Durch einen Klick auf den OK-Knopf (oder [Return]) übernehmen Sie die Änderungen, über CANCEL (oder [Undo]) brechen Sie den Dialog ab, ohne die Änderungen zu übernehmen. Zusatzprogramme wie "CUTLIB", die ebenfalls direkt vom Interpreter aus gestartet werden, sollten auch auf diesem Pfad zu finden sein.

"SAVE SETTINGS"

Speichert verschiedene Einstellungen ab.

Natürlich ist es nicht besonders sinnvoll, sich Editor-Funktionstasten jedesmal auf's neue zu definieren, wenn man das OMIKRON.BASIC startet. Die SAVE SETTINGS-Funktion im Menü MODE speichert die Funktionstasten, das generelle Aussehen des Bildschirms, ob Sie im Insert- oder Overwrite-Modus sind, ob Sie die Zeilennummern an haben oder nicht, ob Fehlerzeilen blinken oder nicht und die Variableneinstellungen (siehe DEFDBL, DEFSNG etc.). Auch der Pfad und die Extension für "Block laden/sichern" wird vermerkt. Diese Einstellungen werden jedesmal zusammen mit OMIKRON.BASIC geladen. Sie sind in einer Datei OM-BASIC.INF enthalten, die normalerweise auf Laufwerk C: im Hauptverzeichnis angelegt wird. Gibt es kein Laufwerk C:, dann wird diese Datei auf dem aktuellen Laufwerk im Hauptverzeichnis angelegt.

Das GO-Menü:

"TO LAST MARK" [Control]-[Z]

Springt zu der Stelle, an der Sie einen Doppelklick gemacht haben, um die LIST-Funktion aufzurufen. Hiermit kommen Sie also immer wieder an Ihre Ausgangsposition zurück, wenn Sie mittels Doppelklick ein Token "verfolgen". (Diese Funktion "merkt" sich bis zu acht Positionen, d.h. Sie können achtmal per Doppelklick suchen und kommen wieder an die Ausgangsposition zurück.)

"GO TO" [F1]

Springt an eine bestimmte Stelle in Ihrem Programm.

Es erscheint eine Dialogbox, in der Sie eine Eingabezeile und sechs Knöpfe finden.

In der Eingabezeile können Sie folgendes eingeben:

Eine Zeilennummer: Der Editor springt in die Zeile, in der die BASIC-Zeile mit dieser Zeilennummer steht.

Einen Offset, (z.B. +5, -8 usw.): Springt um die angegebene Anzahl Programmzeilen vor- bzw. zurück.

Ein Label, (z.B. -TEST): Springt an die Stelle, an der dieses Label steht.

Eine Prozedur, (z.B. P TEST): Springt an die Stelle, an der diese Prozedur definiert ist.

Eine Funktion, (z.B. FN TEST): Springt an die Stelle, an der diese Funktion definiert ist.

Die *Knöpfe* haben folgende Bedeutung:

Block Start: Springt zum Anfang des Blocks.

Block End: Springt zum Ende des Blocks.

Page Top: Verschiebt das Programm so, daß die aktuelle Zeile (in der der Cursor steht) am oberen Bildschirmrand steht.

Page Bottom: Verschiebt das Programm so, daß die aktuelle Zeile am unteren Bildschirmrand steht.

OK oder [Return]: Führt den Sprungbefehl in der Eingabezeile aus.

CANCEL oder [Undo]: Bricht die Funktion ab.

"LINE TO TOP"

Verschiebt das Programm so, daß die aktuelle Zeile am oberen Bildschirmrand steht. (Vgl. "GO TO")

"LINE TO BOTTOM"

Verschiebt das Programm so, daß die aktuelle Zeile am unteren Bildschirmrand steht. (Vgl. "GO TO")

"TO MARK x" bzw. "SET MARK x"

Eine Marke ist eine bestimmte Position in Ihrem Programmtext, die sich der Editor gemerkt hat. Sie können im Fullscreen-Editor 4 verschiedene Marken setzen und wieder anspringen. Diese Funktion ist sehr nützlich, wenn Sie eine Stelle in einem Programm immer wieder anschauen oder editieren müssen. Gemeint sind hier nur die Zifferntasten auf der *Haupttastatur.*

Mit SET MARK X im GO-Menü setzen Sie die Marke X, mit TO MARK X bewegen Sie den Cursor an die gemerkte Stelle X. Die gleichen Funktionen können Sie auch mit [Control]-[<Ziffer>] und [Alternate]-[<Ziffer>] erreichen.

[Control]-[<Ziffer>] setzt eine Marke,

[Alternate]-[<Ziffer>] springt diese Marke wieder an.

"FIND ERROR"

Bringt den Cursor auf den nächsten SYNTAX ERROR oder TYPE MISMATCH ERROR. Logische Fehler können vom Interpreter nicht automatisch erkannt werden.

Das Run-Menü:

"RUN" [Control]-[R]

Verläßt den Editor und startet das im Editor befindliche Programm.

"SAVE & RUN" [Control]-[W]

Speichert das aktuelle Programm unter dem aktuellen Namen (vgl. "SAVE") und startet es anschließend. (Vgl. "RUN") Lautet dieser Name "NONAME.BAS" erscheint beim Speichern eine Dateiauswahlbox.

"TRON & RUN"

Startet das aktuelle Programm und aktiviert zuvor die TRACE-Funktion. (Vgl. auch Befehl TRON und TROFF und Menüpunkt "RUN")

"COMPILE" [Alternate]-[C]

Startet den OMIKRON.BASIC-Compiler unter dem angegebenen Namen und Pfad (vgl. "PREFERENCES") und übersetzt das aktuelle Programm. Das Programm wird dann unter dem aktuellen Namen vom Compiler mit der Endung ".PRG" abgespeichert, sofern es keinen Übersetzer-Fehler gab.

"SAVE & COMPILE" [Alternate]-[Y]

Speichert das aktuelle Programm unter dem aktuellen Namen ab und startet anschließend den Compiler. (Vgl. "SAVE & RUN", "SAVE", "COMPILE")

War der aktuelle Name des Programms "NONAME.BAS", so wird vor dem abspeichern über eine Dateiauswahlbox nach einem neuen Namen gefragt.

"RUN *.BAS"

Startet ein beliebiges BASIC-Programm von Diskette/Festplatte. Zuvor erscheint eine Dateiauswahlbox, in der Sie den Namen des BASIC-Programms auswählen können.

ACHTUNG:

Ihr aktuelles BASIC-Programm

wird bei dieser Funktion

gelöscht.

"EXEC *.PRG" [Control]-[X]

Startet ein beliebiges anderes Programm (z.B. ein Compilat oder ein Textverarbeitungsprogramm...) und kehrt nach Beendigung des anderen Programms in den Editor zurück. Sie sollten aus Sicherheitsgründen vor dem Aktivieren dieser Funktion Ihr momentanes Programm trotzdem abspeichern - für den Fall, daß das aufgerufene Programm abstürzt.

"OS-SHELL"*Nur für Profis!*

Gestattet die Kommunikation mit einer geladenen Shell.

Wenn Sie vor dem Start von OMIKRON.BASIC eine Kommando-Shell (z.B. die Mupfel von Gemini o.ä.) installiert hatten, die Ihren Kommandoprozessor in der Systemvariablen _shell_p ($4F6) eingetragen hat, erscheint

eine Dialogbox in der Sie ein Kommando für diese Shell eingeben können. (z.B. ls, mv, cp, help o.ä.). Haben Sie keine Shell installiert oder unterstützt Ihre Shell die Kommandoübergabe über _shell_p nicht, erscheint eine Dialogbox mit dem Hinweis "No Shell available" (also "Keine Kommandoshell gefunden") und die Funktion wird abgebrochen. Bitte beachten Sie: Wenn Sie diese Funktion verwenden wollen, müssen Sie OMIKRON.BASIC von Ihrer Shell aus starten und nicht über den ATARI-Desktop.

"ACCESSORY"

Wenn Sie auf ACCESSORY klicken, bekommen Sie eine normale GEM-Menüleiste anstatt der vom Editor. Hier können Sie Ihre Accessories verwenden. Mit dem Eintrag "Quit Accessory" können Sie wieder in den Editor zurückkehren.

"HELP"

Ruft das integrierte Hilfesystem auf. Wurde bereits beschrieben.

Weitere Tastaturfunktionen:

Bildschirmvergleich

[Alternate]-[A] zeigt kurzzeitig das letzte Schirmbild des Direktmodus oder des gerade unterbrochenen Programms. Diese Funktion kann nur aufgerufen werden, wenn mit "SCREEN 0" ein graphischer Bildspeicher angefordert wurde (siehe SCREEN).

Einklappen

[Control]-[D] klappt Prozeduren, Funktionen oder einfach nur einen Teil Ihres Programms ein.

Eingeklappt wird bis zum Ende der Procedur oder Funktion oder der eingeklammerte Bereich. Wenn man in einer Zeile die mit "DEF PROC" oder "DEF FN beginnt [Control]-[D] drückt, wird alles was zwischen dieser und der nächsten Zeile, die mit "END_PROC" oder "END_FN" beginnt, eingeklappt. Anschließend wird nur noch die Kopfzeile der betreffenden Prozedur oder Funktion angezeigt. Zum Ausklappen genügt ein erneutes Drücken von [Control]-[D] in der eingeklappten Kopfzeile.

Um andere Teile Ihres Programms einzuklappen, müssen Sie den Bereich durch Setzen zweier geschweiften Klammern ("{" und "}") markieren. Die

Klammer erreichen Sie durch drücken von [Alternate]-[Shift]-[Ö]/[Ä]. Wenn Sie jetzt den Cursor auf die sich öffnende Klammer positionieren und [Control]-[D] drücken, wird alles, was zwischen den Klammer steht, eingeklappt.

Einklappungen mit Klammern sind schachtelbar: Sie können also erst alle Prozeduren einklappen und dann alle eingeklappten Prozeduren nochmals einklappen. Ihr Programm reduziert sich damit auf eine (!) Programmzeile!

Schließlich können Sie eingeklappte Programmteile auch noch gegen das Auflisten (und wieder ausklappen!) schützen. (Vgl. Befehl LOCK).

VORSICHT:

Wenn Sie Ihr Programm im ASCII-Format sichern ("SAVE BLOCK") muß alles *aus*geklappt sein, da sonst nur die Titelzeilen, nicht aber der Inhalt des eingeklappten Programmteils gesichert werden.

[Control]-[Shift]-[D] klappt *alle* eingeklappten Programmteile auf einmal aus.

VORSICHT:

Dieser Befehl läßt sich nicht rückgängig machen - außer durch erneutes *manuelles* Einklappen. Dies kann bei umfangreich verschachtelt eingeklappten Programmen sehr mühsam sein.

Wiederholungsfunktion

[F6] - Repeat

"Repeat" wiederholt die letzte Taste einstellbar oft. Um zum Beispiel die Taste [*] 70mal zu wiederholen (also 70 Sternchen zu erzeugen, z.B. für Überschriftsumrahmungen) tippen Sie:

[*] [F6] [7] [0] [Return]

Sie können auch eine Funktionstaste beliebig oft wiederholen. Das ist besonders bei selbstdefinierten Funktionstasten oft sehr sinnvoll. Um zum Beispiel 10mal die Taste F4 zu drücken, können Sie statt [F4] [F4] [F4] [F4] [F4] [F4] [F4] [F4] [F4] [F4] auch tippen: [F4] [F6] [1] [0] [Return]

Funktionstasten definieren

[Shift]-[F7] - Funktionstaste definieren

Sie können bis zu 64 Zeichen auf eine Funktionstaste legen. Zum Definieren stehen Ihnen die beiden Funktionstasten F4 und F5 zu Verfügung (auch mit Shift - somit haben Sie 4 frei definierbare Funktionstasten). Außerdem können Sie die rechte Maustaste umdefinieren. Sie ist voreingestellt auf BLOCK INSERT.

Beispiel: Sie definieren sich die Funktionstaste F4 mit PRINT ". Das geschieht mit:

[Shift]-[F7] [Shift]-[F4] [P] [R] [I] [N] [T] [ ] ["] [Shift]-[F7]

Ab dann führt jedes Betätigen von [Shift]-[F4] zur Ausgabe von

PRINT "

Weitere Editierhilfen

[F9] - Insert Line

Diese Funktion fügt eine Leerzeile ein.

[Shift]-[F9] - Delete Line

Mit "Delete Line" können Sie eine Zeile ganz löschen. Der Rest des Programm- textes rückt von unten her nach. Besteht eine Programmzeile aus mehereren Bildschirmzeilen, so wird die gesamte Programmzeile gelöscht.

[F10] oder [Control]-[↓] : Seite vor

Blättert eine Bildschirmseite im Programmtext weiter

[Shift]-[F10] oder [Control]-[↑] : Seite zurück

Blättert eine Bildschirmseite im Programmtext zurück

Neben dem Full-Screen-Editor gibt es auch noch einen Bildschirmeditor. Dieser wird auch des öfteren mit "Direktmodus" bezeichnet. Grundsätzlich ist es empfehlenswert, Ihre Programme nur im Full-Screen-Editor einzugeben; zum Testen von Programmen erweist sich allerdings auch der Bildschirmeditor als ein sehr nützliches Werkzeug. Er wird gleich nach den folgenden Tabellen beschrieben.

Abkürzungen für BASIC-Befehle

Einige BASIC-Befehle lassen sich durch Tastenkombinationen mit der [Alternate]-Taste abkürzen. Hier eine Liste dieser Abkürzungen:

[Alternate]-[B]	BLOAD
[Alternate]-[D]	DATA
[Alternate]-[E]	ELSE
[Alternate]-[F]	FOR
[Alternate]-[G]	GOTO
[Alternate]-[H]	HCOPY
[Alternate]-[I]	INPUT
[Alternate]-[K]	KEY
[Alternate]-[L]	LPRINT
[Alternate]-[M]	MID$(
[Alternate]-[N]	NEXT
[Alternate]-[O]	OPEN
[Alternate]-[P]	PRINT
? (ohne [Alternate])	PRINT
[Alternate]-[R]	RETURN
[Alternate]-[S]	SYSTEM
[Alternate]-[T]	THEN
[Alternate]-[U]	USING
[Alternate]-[V]	VARPTR
[Alternate]-[W]	WHILE
[Alternate]-[X]	MOUSEX

Tabellen der Tastaturfunktionen

Tasten im Menü:

[Control]-[B]	Blockstart
[Control]-[C]	Direkt-Mode (Bild schirm-Editor)
[Control]-[E]	Block löschen
[Control]-[F]	Suchen
[Control]-[G]	Weiter suchen
[Control]-[H]	Block verstecken
[Control]-[K]	Blockende
[Control]-[L]	Voreinstellungen
[Control]-[M]	Speichern unter neuem Namen
[Control]-[N]	Neues Programm
[Control]-[O]	Laden ...
[Control]-[P]	Drucken
[Control]-[Shift]-[P]	Block drucken
[Control]-[R]	Programm starten
[Control]-[Q]	Beenden
[Control]-[S]	Speichern
[Control]-[T]	Token suchen
[Control]-[V]	Block verschieben
[Control]-[W]	Speichern und starten
[Control]-[X]	externes Programm starten
[Control]-[Z]	letzte Position
[Help]	Hilfemenü aufrufen
[Control] - [1-4]	Marke setzen
[Alternate]-[1-4]	Marke anspringen
[Alternate]-[C]	Compilieren
[Alternate]-[Y]	Speichern und Compilieren
[F1]	Gehe zu ...
[Shift]-[F1]	Bildschirm teilen
[Shift]-[F2]	Bildschirm wechseln
[Shift]-[F3]	Zeichensatz wechseln

Weitere Tastaturfunktionen

[Control]-[A]	ASCII eingeben
[Alternate]-[A]	Letztes Schirmbild zeigen
[Control]-[D]	Einklappen
[Control]-[Y]	Zeile löschen
[Ctrl]-[Shift]-[Del]	Zeile löschen
[Shift]-[F9]	Zeile löschen
[Ctrl]-[Del]	Rest der Zeile löschen
[F9]	Zeile einfügen
[Ctrl]-[Return]	Zeile auftrennen
[Delete]	Zeichen löschen
[Backspace]	Zeichen links löschen
[Ctrl]-[Insert]	Insert/Overwrite
[F6]	Taste wiederholen
[Shift]-[F7]	Tastendefinition beginnen/beenden
[Tab]	Auf 8er Spalte
[Clr Home]	zum Anfang/Ende
[Control]-[↑]	Seite hoch
[Control]-[↓]	Seite runter
[Control]-[→]	Wort rechts
[Control]-[←]	Wort links

Der Bildschirm-Editor

Nachdem Sie OMIKRON.BASIC gestartet haben, haben Sie automatisch den Full-Screen-Editor vor sich. Um nun in den Bildschirmeditor zu schalten, drücken Sie bitte [Control]-[C] oder klicken den Menüpunkt "DIRECT MODE" im File-Menü an.

Nun haben Sie den Bildschirmeditor vor sich und sehen die Startmeldung mit Versionsnummer ganz oben auf dem Bildschirm, etwas darunter ein OK und nochmal darunter einen blinkenden Cursor. Ansonsten ist der Bildschirm weiß, lädt zum Draufschreiben ein.

Der 'Direktmodus' von OMIKRON.BASIC ist ein vollständiger Bildschirmeditor: Sie können mit dem Cursor beliebig über den Bildschirm fahren und *irgendwo* etwas eintippen, sind also nicht - wie häufig bei einfachen Editoren - zeilengebunden.

Besonderheiten sind:

[Insert] fügt an der aktuellen Cursorposition ein Leerzeichen ein, wobei der Cursor seine Position nicht ändert.

[CTRL]-[↓] Zeile einfügen

[CTRL]-[↑]- Zeile löschen

[CTRL]-[Delete]- Rest der Zeile ab Cursorposition löschen

[CTRL]-[Insert] - Overwrite-Modus einschalten/ausschalten.

Im Bildschirmeditor ist außerdem der volle VT52-Standard integriert. Sie können mit [ESC]-Sequenzen die verschiedensten Funktionen bewirken, z.B. Bildschirm löschen, Rest der Zeile löschen, Cursor nach links oben etc. - die vollständige Tabelle der [ESC]-Sequenzen finden Sie im Anhang. Wenn Sie z.B. (schlagen Sie im Anhang nach) den Rest des Bildschirms löschen wollen, so müssen Sie [ESC] [J] eintippen.

Achten Sie darauf, daß das J ein großes J sein muß, ESC j (mit einem kleinen j) speichert die momentane Cursorposition. An diese gespeicherte Position kommen Sie mit ESC k (kleines k).

Hier noch ein paar weitere Tastenkombinationen im Bildschirm-Editor:

[Alternate]-[CTRL] Tastaturbuffer löschen. Im Tastaturbuffer merkt sich OMIKRON.BASIC Tastendrücke, die es nicht sofort ausführen kann (z.B. wenn Sie Tasten drücken, während ein Programm läuft)

[CTRL]-[Clr] Löscht den Bildschirm

[Return] Merkt die eingegebene Zeile im Programmspeicher (wenn eine Zeilennummer davorsteht) oder führt den eingegebenen Befehl sofort aus (Direktmodus; hier darf keine Zeilennummer am Anfang der Zeile stehen.)

[Control]-[Help] Der Textbildschirm wird neu aufgebaut, so wie OMIKRON.BASIC ihn sich gemerkt hat.

[Shift]-[Shift] Wenn Sie beide Shift-Tasten gleichzeitig gedrückt halten, wird vorübergehend das letzte Schirmbild des Programmes eingeblendet. Damit dies funktioniert muß im Programm ein Graphikbildspeicher mit "SCREEN 1" reserviert worden sein (siehe auch SCREEN).

Neben diesem Bildschirm-Editor gibt es auch noch den bereits beschriebenen Full-Screen-Editor. Wenn Sie im Direktmodus - also dem Bildschirmeditor - arbeiten, erreichen Sie den Full-Screen-Editor einfach mit der **[Help]**-Taste oder mit dem Kommando [E][D][I][T] [<Zeile>]

Wenn Sie vom Full-Screen-Editor wieder in den Bildschirmeditor oder Direktmodus wechseln wollen, drücken Sie [CTRL]-[C] oder klicken den Menüpunkt "DIRECT MODE" im FILE- Menu an.

Arithmetik

In der Anfangszeit der Computer wurden diese nur für Berechnungen eingesetzt. Computer heißt ja auf Deutsch nichts anderes als "Berechner". OMIKRON.BASIC bietet im arithmetischen Bereich besonders viele und auch schnelle Befehle und Funktionen.

Im Computer werden Zahlen in mehreren verschiedenen Zahlformaten dargestellt. Zahlen dieser fünf verschiedenen Formate können in den fünf entsprechenden Variablentypen gespeichert werden:

INTEGER

Integerzahlen sind *ganze* Zahlen, also Zahlen ohne Nachkonmmastellen. Ihr Zahlenbereich ist eingeschränkt auf die unten angegebenen Werte. Vorteile dieser Zahlart sind:

- Sehr schnelle Abarbeitung
- Es gibt keine Rundungsfehler

Long-Integer:	Rechenbereich: -2147483658 bis +2147483657
	Speicherplatzbedarf: 4 Byte (=4 Zeichen)
	Variablennamen: **A%L, A%L(1)** (Postfix: **%L**)
Integer word:	Rechenbereich: -32768 bis +32767
	Speicherplatzbedarf: 2 Byte (=2 Zeichen)
	Variablennamen: **A%, A%(1)** (Postfix: %)
Integer byte:	Rechenbereich: 0 bis 255
	Speicherplatzbedarf: 1 Byte (=1 Zeichen)
	Variablennamen: **A%B(1)** (Postfix: **%B**)
Flag/Boolean:	Rechenbereich: "falsch" (0) und "wahr" (-1)
	Speicherplatzbedarf: 1 Bit (= ein achtel Zeichen)
	Variablennamen: **A%F(1)** (Postfix: **%F**)

FLOAT (Fließkomma)

Fließkommazahlen sind Zahlen mit Nachkommastellen. Ihr Zahlenbereich ist (fast) gar nicht begrenzt, da sie einen Zehnerexponenten besitzen: Reicht der Zahlenbereich der Mantisse (s.u.) nicht aus, so wird der Exponent jeweils um eins erhöht, und die Mantisse wird um eine Stelle nach rechts geschoben.

Dadurch gehen jedoch immer mehr Nachkommastellen verloren, je größer die Zahl ist. Die gesamte Stellenzahl bleibt dagegen erhalten:

123,45678 mal 10 hoch 2

Mantisse Exponent

Vorteile dieser Zahlart sind:

- Nachkommastellen
- Die Genauigkeit paßt sich der Größenordnung der Zahl an.

single-Float:	Rechenbereich: +/- 5,11 mal 10 hoch 4931
(auch: short-	Genauigkeit: 9,5 Stellen
Float)	Speicherplatzbedarf: 6 Byte (=6 Zeichen)
	Variablennamen: **A!**, **A!(1)** (Postfix: !)
double-Float:	Rechenbereich: +/- 5,11 mal 10 hoch 4931
(auch: long-	Genauigkeit: 19 Stellen
Float)	Speicherplatzbedarf: 10 Byte (=10 Zeichen)
	Variablennamen: **A#**, **A#(1)** (Postfix: #)

STRING (Zeichenkette)

Ist hier der Vollständigkeit halber aufgeführt (es ist kein *Zahl*-, dafür aber ein *Variablen*- Typ): Eine Zeichenkette ist ein Text von bis zu 32766 Zeichen Länge. Der Text kann nicht nur druckbare Zeichen enthalten (Buchstaben, Ziffern, Sonderzeichen), sondern auch nichtdruckbare und Steuerzeichen. (Siehe Anhang: ASCII-Tabelle)

String:Länge: 0 bis 32766 Zeichen

Speicherplatzbedarf: 16 Bytes (=16 Zeichen) plus Länge

Variablennamen: **A$**, **A$(1)** (Postfix: $)

Wichtig: Die Variablentypen Flag und Integer-Byte können *ausschließlich* in Variablenfeldern verwendet werden.

Zu den Variablennamen: Die Variable **A** soll nur ein Beispiel sein. Grundsätzlich kann ein Variablen-Name aus bis zu 31 Buchstaben, Ziffern und Unter- streichungszeichen (neben rechter [Shift]-Taste) bestehen. Das erste Zeichen davon *muß* allerdings ein Buchstabe sein.

Oben wurde -zig mal die Variable **A** als Beispiel für Variablennamen genannt. OMIKRON.BASIC unterscheidet *jede einzelne* Variable voneinander, obwohl sie alle **A** heißen. Durch die Postfixe für die einzelnen

Variablentypen unter- scheiden sie sich bereits. Wenn Sie die Variable **A%L** bereits verwendet haben, können Sie sogar noch ein davon unabhängiges Feld **A%L(1)** und eines **A%L(1,2)** verwenden. Hier besteht nämlich der Unterschied in der Anzahl der Parameter. Es ist jedoch besser, solche Namensgleichheiten zu vermeiden, sonst kann ein Programm schnell unübersichtlich werden.

Sie können übrigens für jeden Buchstaben des Alphabets einen Standard-Variablentyp vergeben. Jede Variable, die mit ebendiesem Buchstaben beginnt, hat dann automatisch den Standard-Variablentyp - es sei denn, sie geben das Postfix eines anderen Variablentyps an:

nach DEFINT A-Z	sind *alle* Variablen, die Sie ohne Postfix eingeben, automatisch vom Typ **Integer word**.
nach DEFINTL I	sind alle Variablen, die mit I anfangen und die Sie ohne Postfix eingeben, automatisch vom Typ **long-Integer**.
nach DEFSNG A-Z	sind *alle* Variablen, die Sie ohne Postfix eingeben, automatisch vom Typ **single-Float**.
nach DEFDBL A-Z	sind *alle* Variablen, die Sie ohne Postfix eingeben, automatisch vom Typ **double-Float**.
nach DEFSTR S,U-V	sind alle Variablen, die mit S, U oder V beginnen und die Sie ohne Postfix eingeben, automatisch vom Typ **String**.

Beim Ausgeben des Programmes (mit dem Befehl **LIST**) wird bei Variablen, die vom Standard-Variablentyp sind, das jeweilige Postfix weggelassen.

Die Einstellung direkt nach dem Laden des Interpreters ist:

DEFINTL A-Z

Das bedeutet, daß Sie long-Integer-Variablen eingeben können, *ohne* das **%L** jedesmal hintendranzutippen. Wenn bei Beispielen im Handbuch also kein Variablentyp angegeben ist, so sind stets Long-Integer-Variablen gemeint.

Falls Sie bereits andere BASIC-Interpreter kennen, werden Sie sich sicherlich darüber wundern, daß eine normale Variable in OMIKRON. BASIC vom Typ long-Integer ist und nicht vom Typ single-Float. Der Grund mag vielleicht ein Stück Idealismus sein. Wir haben uns nämlich *deshalb* dazu entschlossen, weil:

- in den meisten Fälle, in denen Variablen verwendet werden, keine Fließkommavariablen notwendig sind,
- Integer-Berechnungen nun mal wesentlich schneller sind,
- Integer-Berechnungen von einem Compiler besser übersetzbar sind (Der OMIKRON.Compiler bringt etwa Faktor 2,5, bei Fließkomma-Grundrechenarten dagegen Faktor 8-10 bei Integer-Anwendungen!),
- und weil man durch Eingeben von DEFSNG A-Z wieder den von anderen Interpretern her bekannten Fließkomma-Standard hat.

Ach ja - noch etwas zu DEFSNG und DEFINTL und den anderen: Ist erst einmal eine Variable vom Typ long-Integer eingegeben worden, so hilft kein nachträgliches DEFSNG: Die Variable behält Ihren Typ bei! (Sie wird dann aber von LIST als A%L statt als A aufgeführt, während eine zweite Variable A! nunmehr als A angezeigt wird.)

Wenn Sie also Programme, die auf anderen Rechnern entwickelt wurden, übernehmen wollen, so geben Sie ein:

```
DEFSNG A-Z
OK
LOAD "VONMBASI"    (der Name Ihres Programmes)

OK
1 DEFSNG A-Z       (damit die Definition dann auch im Programm steht)
```

Konstanten

Konstanten sind Zahlen, die sich nicht verändern:

123	(Integer)
1.3	(Fließkomma)
1E20	(Fließkomma)
123!	(Fließkomma)
123#	(Fließkomma; doppelt genau)
1D20	(Fließkomma; doppelt genau)
1.23456789	(Fließkomma; doppelt genau)

der Vollständigkeit halber (es ist keine *Zahl*- Konstante):

"hallo" (String)

Eine Zahl, die eines der folgenden Zeichen enthält, ist automatisch Fließkommazahl:

. ! # E D

Die Postfixe ! und # können also auch hinter Zahlen stehen.

Der Buchstabe E steht für "Zehnerexponent", D für den Zehnerexponent einer doppelt genauen Zahl.

Sowohl der Buchstabe D als auch das # erzwingen eine doppelt genaue Zahl. Ebenso ist eine Zahl mit mehr als acht gültigen Stellen automatisch doppelt genau.

Probieren Sie doch bitte einmal:

```
PRINT CDBL(.1)
```

CDBL wandelt die einfach genaue Konstante 0,1 in doppelte Genauigkeit. Dennoch ist das Ergebnis nur 9 Stellen genau.

Wir erkennen daraus: Will man ein doppelt genaues Ergebnis haben, müssen die Konstanten, die in einer Berechnung verwendet werden, auch doppelt genau sein.

Die Berechnung

```
A#=1/3:PRINT A#
```

ist also ungenau. Richtig ist:

```
A#=1#/3#:PRINT A#
```

- und das Ergebnis stimmt!

Wie Sie jetzt wissen, kann man in Integer keine Nachkommastellen darstellen, außerdem ist der Zahlenbereich begrenzt. Für viele arithmetischen Funktionen werden aber Nachkommastellen benötigt (z.B.: Sinus), so daß die nun folgenden Abschnitte sich oft auf Fließkommazahlen beziehen. Selbstverständlich läßt sich jede dieser Funktionen auch in doppelter Genauigkeit berechnen.

Für Profis: "Single precision" oder "einfache Genauigkeit" bedeutet ein Speicher- und Rechenformat von 6 Bytes pro Zahl. Davon sind 33bit (32+Vorzeichen) für die Mantisse reserviert, 15 bit für den Exponenten. OMIKRON.BASIC erreicht hier eine Genauigkeit von 2*1E-10, also etwas mehr als 9 gültige Stellen. Davon werden lediglich 8 ausgegeben, der Rest wird aber mitgerechnet und mitabgespeichert.

Wenn diese Genauigkeit nicht ausreicht, muß man in doppelter Genauigkeit rechnen.

"Double precision" bedeutet ein Format von 10 Bytes pro Zahl. Der Speichermehrverbrauch macht sich vor allem bei Arrays und Matrizen mit double-Zahlen bemerkbar. Die 10 Byte teilen sich in 65bit Mantisse und 15bit Exponent auf. Damit ergibt sich derselbe Rechenbereich wie mit single-Zahlen, aber eine Genauigkeit von 5E-20, also über 19 gültige Stellen, von denen 17 ausgegeben werden.

Die erhöhte Genauigkeit bei double-Zahlen bezahlt der Anwender mit höherem Speicherplatzbedarf und geringerer Verarbeitungsgeschwindigkeit (4- bis 8-fache Rechenzeit).

Rundungsfehler

Leider kann man beim Rechnen mit Floatzahlen Rundungsfehler nicht vermeiden. Das liegt nicht daran, daß unser Programmierer sein Metier nicht beherrscht, sondern es ist eine prinzipielle Schwäche von Zahlendarstellungen mit endlich viel abgespeicherten Stellen. Ein Beispiel aus dem täglichen Leben verdeutlicht dies:

1/3 = 0.33333333

Um die Zahl "1/3" darzustellen, können Sie beliebig viele Dezimalstellen angeben. Die angegebene Zahl wird jedoch nie genau ein Drittel sein, weil Sie irgendwann abbrechen müssen, und die Zahl ja noch mit unendlich vielen Dreiern weitergeht.

Intern speichert der Computer alle Zahlen im Zweier- bzw. Dualsystem ab. Im Zweiersystem kann man aber Zahlen, die in unserem normalen Zehnersystem des täglichen Lebens keine Probleme machen, nicht darstellen. Ein Beispiel dafür stellt die Zahl 0.1 dar. Probieren Sie einmal:

```
PRINT 10000-10000.1
```

Das Ergebnis dürfte Sie überraschen. Aber probieren Sie es ruhig mal auf einem anderen Computer (keinem Taschenrechner, denn die rechnen im Dezimalsystem) aus!

Um Rundungsfehler möglichst gering zu halten, sollten Sie Subtraktionen von großen, ähnlichen Zahlen (wie im obigen Beispiel) vermeiden.

Übersicht

Befehle, Funktionen, Operatoren

Zur Syntaxerklärung:

Angaben in spitzen Klammern <...> sind durch entsprechende Ausdrücke zu ersetzen.

Angaben in eckigen Klammern [...] sind, je nach gewünschter Anwendung, wegzulassende Ergänzungen.

Sind um einen Ausdruck zwei eckige Klammern [[...]] gesetzt, so kann, je nach Anwendung, dieser Ausdruck weggelassen werden, einmal oder mehrmals erscheinen.

Angabe in geschweiften Klammern {...|...} stellen eine Auswahl dar, aus der eine der durch | getrennten Alternativen auszuwählen ist.

Zu den Beispielen:

Beispielprogramme, die so abgetippt werden können, sind mit Zeilennummern abgedruckt.

Bildschirmergebnisse, sofern es sich nicht um Grafiken etc. handelt, sind durch eine Leerzeile getrennt aufgeführt.

Steht keine Zeilennummer vor dem Befehl, so führt die Ausführung dieses Befehls in dieser Form zu keinem vernünftigen Ergebnis. Das Beispiel dient dann nur zur Verdeutlichung der Syntax!

Weitere Erklärungen:

<num.Ausdruck>

Ein numerischer Ausdruck ist ein beliebiger Term, der ein numerisches Ergebnis liefert. Dabei spielt der Typ (ganzzahlig, einfach- oder doppeltgenau) keine Rolle.

<Stringausdruck>

Ein Ausdruck vom Typ String. Er kann insbesondere auch die Funktion @(<Zeile>, <Spalte>) enthalten.

<Funktionsnummer>

Vorzeichenlose ganze Zahl, die eine Funktion aus den Betriebssystem Bibliotheken AES, BIOS, GEMDOS, VDI oder XBIOS auswählt.

<GLOBAL-Feld>

Globales Parameterfeld für das AES. Hier steht der angegebene Index für die Zahl der im Feld enthaltenen Werte.

<INTIN-Feld>

Globales Parameterfeld für das AES und das VDI. Hier steht der angegebene Index für die Zahl der im Feld enthaltenen Werte.

<INTOUT-Feld>

Globales Parameterfeld für das AES und das VDI. Hier steht der angegebene Index für die Zahl der im Feld enthaltenen Werte.

<ADDRIN-Feld>

Globales Parameterfeld für das AES. Hier steht der angegebene Index für die Zahl der im Feld enthaltenen Werte.

<ADDROUT-Feld>

Globales Parameterfeld für das AES. Hier steht der angegebene Index für die Zahl der im Feld enthaltenen Werte.

<Dateiname>

Ein Dateiname ist ein String, der folgenden Aufbau hat:

```
"[‹Laufwerk›][\][[‹Ordnername›\]]‹Name›"
```

Laufwerk steht für eine das Laufwerk kennzeichnenden Buchstaben zwischen "A" und "P", gefolgt von einem Doppelpunkt.

Ein Ordnername besteht aus bis zu acht Zeichen und eventuell einer drei Zeichen langen Namenserweiterung, die durch "." abgetrennt wird. Der Ordnername darf keine Jokerzeichen beinhalten. Ein aus einem oder mehreren Ordnern bestehder Dateiname nennt man auch Pfad. Der Pfad heißt vollständig, wenn angefangen vom Laufwerk bis hin zum Namen alle Angaben vorhanden sind.

Der Name selbst entspricht dem Ordnernamen, nur daß hier fast immer auch Jokerzeichen zugelassen sind.

Wenn bei der Angabe des Dateinamens die Laufwerksangabe fehlt, so wird vom Standardlaufwerk ausgegangen. Fehlt auch der Pfad so wird der Standardpfad verwendet.

<num. Variable>

Numerische Variable gleich welchen Typs. In Einzelfällen sind möglicherweise nur einfach Variablen zugelassen und keine Felder.

<Rückgabe-Variable>

Liefert den Ergebniswert bzw. die Fehlernummer einer Betriebssystemfunktion zurück. Bei INPUT USING Art der Abbruchbedingung.

<Parameter>

Die an eine Funktion oder Prozedur übergebenen Werte. Bei Betriebssystemaufrufen des BIOS, GEMDOS, XBIOS oder bei Aufrufen mit CALL kann durch ein vorangestelltes "L " eine Übergabe als LONG-WORD bewirkt werden.

<Bitnummer>

Vorzeichenlose Zahl zwischen 0 und 31. Die Wertigkeit des entsprechenden Bits ist 2^<Bitnummer>.

<Integer-Variable>

Ganzzahlige Variable WORD oder LONG, auch Feldvariablen sind zulässig.

<Speicheradresse>

32 Bit breite (1 LONG) Integer-Zahl, die eine bestimmte Speicherzelle benennt.

<Marke>

Eine Marke bezeichnet eine bestimmte Stelle eines Programms:

1. Eine bestimmte Zeile. Wenn mit Zeilennummern gearbeitet wird kann man durch Nennung der Nummer eine bestimmte Zeile erreichen. Die Zeilennummer kann auch berechenet werden, d.h. als Marke ist auch ein beliebiger numerischer Ausdruck zugelassen. Wenn der Ausdruck jedoch aus einer einzigen numerischen Variablen besteht, so ist diese in Klammern zu setzen, um nicht mit einem Label verwechselt zu werden.
2. Ein bestimmtes Programm-Label: An jeder Stelle des Programms kann mit "-<Bezeichner>" ein Label vereinbart werden. Der Bezeichner darf nur die üblichen Zeichen enthalten (alle Buchstaben, Ziffern und "_", aber z.B. keine Umlaute). Will man sich auf ein solches Label beziehen wird einfach der Bezeichner genannt, das vorangestellte Minuszeichen unterbleibt. Man kann stattdessen, das Label auch über einen Stringausdruck benennen. Er muß den Bezeichner des Label in Großbuchstaben enthalten und darf für Compilate nicht länger als acht Zeichen sein.

<Laufwerk>

Laufwerk steht für eine das Laufwerk kennzeichnenden Buchstaben zwischen "A" und "P", gefolgt von einem Doppelpunkt.

<Winkel>

Winkelangaben sind stets in 1/10 Grad vorzunehmen, d.h. z.B. 900 entspricht 90 Grad.

<Dateinummer>

Eine vorzeichenlose Zahl zwischen 1 und 16. Die Zahl steht in fester Verbindung mit der zugehörigen Datei (Siehe OPEN). Alle Befehle und Funktionen wirken auf die Datei, die der Dateinummer durch OPEN zugeordnet wurde.

Rechenzeichen

Zeichen	*Bedeutung*
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
^	Potenzieren
+=	Addition mit gleichzeitiger Zuweisung des Ergebnisses
-=	Subtraktion "
*=	Multiplikation "
/=	Division "
\	Integer-Division (mit ganzzahligem Ergebnis !!!)
	Vorteil der Integer-Division ist, daß sie erheblich schneller abläuft als eine Division mit Fließkomma. Wenn Sie also den Quotienten zweier integer-Variablenberechnen wollen (und/oder nur den ganzzahligen Anteil des Ergebnisses benötigen, sollten Sie \ verwenden.
MOD	Modulo-Operator: Liefert den Rest einer Ganzzahl-Division

Beispiele:

```
A+=5  ist gleichbedeutend mit A=A+5
C-=8  --> C=C-8
E*=3  --> E=E*3
G/=10 --> G=G/10
```

Diese vier Operatoren mit gleichzeitiger Zuweisung wurden der Programmiersprache C entlehnt. Wer sich also bereits in C auskennt, wird keinerlei Probleme haben, diese Operatoren zu verwenden. Der Vorteil dieser Operatoren wird besonders dann deutlich, wenn komplizierte Ausdrücke berechnet werden müssen. Besonders im Compilat ergibt sich dadurch ein gewaltiger Geschwindigkeitsvorteil.

```
PRINT 2^5, 3002\1000, 3002 MOD 1000
32     3      2
OK
```

Liste der Operatoren nach Priorität:

(,)	(höchste Priorität)
NOT	.
^ (Potenzieren)	.
SHL, SHR	.
*, /	
\, MOD	
+, -, - (Vorzeichen)	
<, <=, >, >=, <>	
AND	
OR	
EQV, IMP, NAND, NOR, XOR	(niedrigste Priorität)

Beispiele:

```
PRINT -3^2, (-3)^2, 3*1 SHL 2, (3*1) SHL 2
-9          9          12          18
OK

PRINT -1>=2 OR 1<2, -1>=((2 OR 1)<2)
-1          0
OK
```

Vergleichsoperatoren

OMIKRON.BASIC stellt die folgenden Vergleichsoperatoren zur Verfügung:

Zeichen	*Bedeutung*
>	größer als
<	kleiner als
=	gleich
>=	größer oder gleich
<=	kleiner oder gleich
<>	ungleich

Pointer/Adressoperatoren

"&" ermittelt die Adresse einer Variablen, Funktion oder eines Feldes. Zurückgegeben wird die Adresse des Objekts (Variable, Funktion, Feld) im Speicher. Da zumindest im Interpreter nur relative Zeiger auf das jeweilige Segment (vgl. SEGPTR/VARPTR) zurückgegeben werden, können Sie nicht direkt auf die Objekte über diesem Wert zugreifen, sondern müssen hierfür den Dereferenz-Operator "*" verwenden. Der Adreß- Operator "&" kann insbesondere auch Zeiger auf Funktionen ermitteln und somit können Funktionen indirekt aufgerufen werden.

"*" greift auf ein Objekt zu (dereferenziert es). Direkt hinter dem Operator muß die Zeigervariable folgen. Welcher Typ gemeint ist, muß auch hier - wie bei einer normalen Variablen - noch mit einem Postfix angegeben werden. Speziell wenn es sich um einen Zeiger auf Funktionen handelt, ruft "*" die Funktion auf die die Zeigervariable zeigt auf.

Beispiel:
```
0 Text$="Dies ist ein Beispieltext"
1 Ptr_Text=&Text$
2 PRINT *Ptr_Text$ 'Das $-Zeichen ist wichtig!

Dies ist ein Beispieltext
```

Beispiel:
```
0 Summe=0
1 Ptr_Summe=&Summe
2 N=10
3 FOR I= 1 TO N
4   *Ptr_Sum=*Ptr_Sum+I*I
5 NEXT I
6 PRINT Summe
```

Zeiger auf Funktionen

Der Adreß-Operator "&" ermittelt die Adresse eines Objekts (Variable, Feld oder Funktion). Der Ausdruck: "Ptr_Str_Cmp=&FN Cmp(A$, B$)" speichert in der Variable "Ptr_Str_Cmp" einen Zeiger auf die Funktion "FN Cmp(,)" (also deren Adresse) ab. Mit diesem Zeiger kann die Funktion nun indirekt aufgerufen werden: "PRINT FN *Ptr_Str_Cmp("A","B") " Solche Funktions- Zeiger können ebenso wie Zeiger auf ein Feld an Prozeduren übergeben werden. Mit oberem Beispiel wäre eine "Sortier-Prozedure" denkbar, die zum String-Vergleich lediglich einen Funktionszeiger übergeben bekommt. Abhängig von der Vergleichsfunktion wird nach ganz unterschiedlichen Kriterien sortiert (z.B. aufsteigend, absteigend oder groß=klein ...)

Beispiel: (auch im DEMO-Ordner)

```
Ptr=&FN Sin2#(X#)
PRINT "Wertetabelle von Sinusquadrat:"
Werte_Tab Ptr,0#,1#,.1#
PRINT "Wertetabelle von Cosinusquadrat:"
Werte_Tab &FN Cos2#(X#),0#,1#,.1#
'
A$= INPUT$(1)
END
'
DEF FN Sin2#(X#)= SIN(X#)* SIN(X#)
DEF FN Cos2#(X#)= COS(X#)* COS(X#)
'
DEF PROC Werte_Tab(Ptr_To_Df_Fn,Von#,Bis#,Schritt#)
   LOCAL X#=Von#
   '
WHILE X#<=Bis#
   PRINT X#,FN *Ptr_To_Df_Fn#(X#)  'Vorsicht:
   X#=X#+Schritt#                  'Hier kann compiliert
WEND                               'keine Typprüfung mehr
RETURN                             'durchgeführt werden
```

WICHTIG: Wenn Sie auch den Compiler verwenden, achten Sie auf genaue Übereinstimmung der Parameterlisten der über Zeiger aufgerufenen Funtionen. Das Compilat kann zur Laufzeit keine Typanpassung vornehmen, da nicht bekannt ist auf welche Funktion der Zeiger gerade verweist. Im oberen Beispiel dürfen Sie "FN *Ptr_To_Df_Fn" nur mit einem Double-Float als Parameter aufrufen (hier X#). Die normalerweise notwendige Konvertierung von integer nach double-float kann nicht automatisch erfolgen, der Integer-Parameter wird falsch übergeben.

Weitere Beispiele finden Sie im Ordner DEMO auf der Programmdiskette.

ABS

Typ: Funktion

Syntax: ABS(<num.Ausdruck>)

Erklärung: ABS berechnet den Absolutbetrag des numerischen Ausdrucks. Negative Werte erhalten so ein positives Vorzeichen, während positive Werte unverändert bleiben.

Beispiel:

```
0 A=-5.2
1 PRINT A
2 PRINT ABS(A)
3 PRINT ABS(A+3.2)+3.2

-5.2
 5.2
 5.2
```

AES

Typ: Befehl

Syntax: AES(<num.Ausdruck>, <Feldvariable>, <Feldvariable>, <Feldvariable>, <Feldvariable>, <Feldvariable>)

AES(<Funktionsnummer>, <GLOBAL-Feld>, <INTIN-Feld>, <ADDRIN-Feld>, <INTOUT-Feld>, <ADDROUT-Feld>)

Erklärung: Mit dem Befehl AES rufen Sie GEM AES (Graphics Environment Manager - Application Environment Services) auf. Die einzelnen Funktionen des GEM AES schlagen Sie bitte im Anhang nach.

Für VDI- und AES-Befehle benutzen Sie einfacher die GEM- oder EasyGEM-Library.

AND

Typ: Operator

Syntax: <num.Ausdruck> AND <num.Ausdruck>

Erklärung: Die beiden Ausdrücke werden bitweise "logisch und" verknüpft.

Beispiel:

```
0 PRINT BIN$(%1010 AND %1100)
1 IF 1<2 AND 2<3 THEN PRINT "Hab' ich's doch gewußt!"
2 IF 1<4 AND 2=4 THEN PRINT "Das ist aber falsch!!!"
```

```
1000
Hab' ich's doch gewußt!
```

ARCCOS

Typ: Funktion

Syntax: ARCCOS(<num.Ausdruck>)

Erklärung: Berechnet den arcus cosinus des numerischen Ausdrucks. Der numerische Ausdruck muß einen Wert zwischen -1 und 1 sein. Das Ergebnis ist abhängig vom eingestellten Winkelmodus (siehe DEG, RAD).

Beispiel:

```
0 PRINT ARCCOS(-.5)
1 DEG
2 PRINT ARCCOS(1)

 2.0943951
 84.26083
```

ARCCOT

Typ: Funktion

Syntax: ARCCOT(<num.Ausdruck>)

Erklärung: Berechnet den arcus cotangens des numerischen Ausdrucks. Das Ergebnis ist abhängig vom eingestellten Winkelmodus (siehe DEG, RAD).

Beispiel:

```
0 PRINT ARCCOT(5),
1 DEG
2 PRINT ARCCOT(1)

 .19739556          45
```

ARCOTH

Typ: Funktion

Syntax: ARCOTH(<num.Ausdruck>)

Erklärung: Berechnet den area cotangens hyperbolicus des numerischen Ausdrucks. Der numerische Wert muß außerhalb des Bereichs von -1 bis 1 liegen.

Beispiel:

```
Ø PRINT ARCOTH(5)

 .20273255
```

ARCSIN

Typ: Funktion

Syntax: ARCSIN(<num.Ausdruck>)

Erklärung: Berechnet den arcus sinus des numerischen Ausdrucks. Der numerische Ausdruck muß ein Wert zwischen -1 und 1 sein. Das Ergebnis ist abhängig vom eingestellten Winkelmodus (siehe DEG, RAD).

Beispiel:

```
Ø PRINT ARCSIN(1),
1 DEG
2 PRINT ARCSIN(.5)

 1.5707963       30
```

ARCTAN

Typ: Funktion

Syntax: ARCTAN(<num.Ausdruck>)

Erklärung: Berechnet den arcus tangens des numerischen Ausdrucks. ARCTAN ist identisch mit ATN. Das Ergebnis ist abhängig vom eingestellten Winkelmodus (siehe DEG, RAD).

```
   INT ARCTAN(-1)
1 DEG
2 PRINT ARCTAN(1)

-.78539816
 45
```

ARSINH

Typ: Funktion

Syntax: ARSINH(<num.Ausdruck>)

Erklärung: Berechnet den area sinus hyperbolicus des numerischen Ausdrucks.

Beispiel:
```
Ø PRINT ARSINH(-1)

-.88137359
```

ARTANH

Typ: Funktion

Syntax: ARTANH(<num.Ausdruck>)

Erklärung: Berechnet den area tangens hyperbolicus des numerischen Ausdrucks. Der numerische Ausdruck muß ein Wert innerhalb von -1 und 1 sein.

Beispiel:
```
Ø PRINT ARTANH(.5)
-.54930614
```

AS

Erklärung: siehe NAME ... AS

ASC

Typ: Funktion

Syntax: ASC(<Stringausdruck>)

Erklärung: Ermittelt den ASCII-Wert des ersten Zeichens vom Stringausdruck. ASC ist die Umkehrfunktion von CHR$.

Siehe auch ASCII-Tabelle. Achtung: ASC("") ergibt eine Fehlermeldung. Der Stringausdruck darf kein Leerstring sein.

Beispiel:
```
Ø PRINT ASC("abc")
1 PRINT ASC( CHR$(24))

 97
 24
```

ATN

Typ: Funktion

Syntax: ATN(<num.Ausdruck>)

Erklärung: Berechnet den arcus tangens des numerischen Ausdrucks. ATN ist identisch mit ARCTAN. Das Ergebnis ist abhängig vom eingestellten Winkelmodus (siehe DEG, RAD).

Beispiel:
```
0 PRINT ATN(-1)
1 DEG
2 PRINT ATN(1)

-.78539816
 45
```

BACKUP

Typ: Befehl

Syntax: BACKUP <Stringausdruck>

BACKUP <Dateiname>

Erklärung: Erstellt von der/den Datei(en) eine Sicherheitskopie.

Die Sicherheitskopie erhält die Extension ".BAK" und wird im selben Verzeichnis abgelegt. Jokerzeichen im Dateinamen sind zugelassen.

Kann/können die Datei(en) nicht gefunden werden, wird der Befehl abgebrochen und ohne Fehlermeldung im Programmablauf fortgefahren.

Beispiel:
```
BACKUP "a:\*.BAS"
BACKUP "version?.*"
```

BIN$

Typ: Funktion

Syntax: BIN$(<num.Ausdruck>)

Erklärung: Wandelt den numerischen Ausdruck in eine Zeichenkette um, die den gerundeten Wert des Ausdrucks als Binärzahl darstellt. Der num. Ausdruck wird immer zuvor nach Integer-Langwort-Format gewandelt. Ein betragsmäßig zu großer Wert erzeugt ein "Overflow".

Beispiel:
```
0 PRINT BIN$(4),BIN$(-3)
1 PRINT BIN$(24)
2 PRINT BIN$(1E+10)

 100        -11
```

```
 11000
? Overflow in 2
```

BIOS

Typ: Befehl

Syntax: BIOS[()][<num. Variable>],<num.Ausdruck>

[[,[L]<num.Ausdruck>]][()]

BIOS[()][<Rückgabe-Variable>],<Funktionsnummer> [[,[L]<Parameter>]][()]

Erklärung: Die in Funktionsnummer genannte BIOS-Funktion wird aufgerufen und Parameterliste übergeben. Der Rückgabe-Wert der BIOS-Funktion wird der Rückgabe-Variable zugewiesen. Wenn vor den Parametern ein "L " gestellt ist, so wird der Parameter als LONG übergeben, ansonsten immer als WORD.

Siehe auch BIOS-Funktionsliste.

Beispiel:

```
0-Schleife
1 BIOS(Shift,11,-1)
2 PRINT @(0,0);Shift
3 GOTO Schleife

 4, wenn CONTROL gedrückt
```

BIT

Typ: Funktion

Syntax: BIT(<num.Ausdruck>,<num.Ausdruck>)

BIT(<Bitnummer>,<Wert>)

Erklärung: Der gerundete Wert wird in eine 32 stellige Binärzahl gewandelt (wie CINTL). Ein betragsmäßig zu großer Wert erzeugt ein "Overflow".

Die Funktion ergibt -1, wenn das von rechts gezählte Bit der Binärarzahl gesetzt ist, andernfalls 0.

Das niederwertigste Bit hat die Bitnummer 0.

Das höchstwertigste Bit hat die Bitnummer 31 und stellt das Vorzeichen dar. Ein falsche Bitnummer erzeugt ein "Illegal function call".

Beispiel:

```
0 PRINT BIT(0,1)
1 PRINT BIT(2,1)
2 PRINT BIT(31,-1)
```

```
3 PRINT BIT(12,-1)

-1
 0
-1
-1
```

BIT

Typ: Befehl

Syntax: BIT (<num.Ausdruck>,{<Integer- Variable>|(<num.Ausdruck>)})=<num. Ausdruck>

BIT (<Bitnummer>,{<Integer-Variable>|(<Speicheradresse>)}=<Bit-wert>)

Erklärung: Weist, wenn Bitwert null ist, einem einzelnen Bit den Wert null zu, andernfalls den Wert eins.

Das Bit wird durch Bitnummer angegeben. Das niederwertigste Bit hat die Bitnummer null.

Es können entweder Bits von Ganzzahl-Variablen oder von Bytes im Speicher geändert werden.

WICHTIG: Bei Änderungen von Speicher erfolgt der Zugriff immer byteweise d.h. die höchste Bitnummer ist 7. Beim Modifizieren von Integer-Variablen hingegen sind Bitnummer bis 31 erlaubt. Eine falsche Bitnummer führt zu einem "Illegal function call".

Beispiel:

```
0 Adresse=MEMORY(1)
1 PRINT PEEK(Adresse)
2 BIT (2,(Adresse))=1
3 PRINT PEEK(Adresse)
4 A=19
5 BIT (2,A)=0
6 PRINT A

 0
 4
 17
```

BITBLT

Typ: Befehl

Syntax: BITBLT <num.Ausdruck>[,<num.Ausdruck>,<num.Ausdruck>,<num. Ausdruck>] TO <num.Ausdruck>[,<num.Ausdruck>,<num. Ausdruck>, <num.Ausdruck>][;<num.Ausdruck>]

BITBLT <num.Ausdruck> TO <num.Ausdruck>, COLOR <num.Ausdruck>

1: BITBLT <X1>,<Y1>,<Breite1>,<Höhe1> TO <X2>,<Y2>, <Breite2>, <Höhe2>[;<Modus>]

2: BITBLT <X>,<Y>,<Breite>,<Höhe>TO <Speicheradresse>[;<Modus>]

3: BITBLT <Speicheradresse> TO <X>,<Y>,<Breite>,<Höhe>[;<Modus>]

4: BITBLT <Speicheradresse1> TO <Speicheradresse2>,COLOR <Farbe>

Erklärung: Kopiert einen rechteckigen Bildschirmausschnitt. Als Quelle bzw. Ziel dienen je nach Syntax:

1: von Bildschirm nach Bildschirm

2: von Bildschirm nach Speicher

3: von Speicher nach Bildschirm

4: von Speicher nach Speicher

Syntax 4 ermöglicht die Umwandlung von s/w-Vorlagen in Farbblits, die dann direkt in den Bildschirm geblittet werden können. Als Farbe ist je nach Auflösung ein Wert zwischen 0 und 1, 0 und 3, bzw. 0 und 15 anzugeben (0 bis 255 bei ATARI TT).

Bei Syntax 1 bis 3 ist zusätzlich die Angabe eines Modus möglich, der eine punktweise Verknüpfung der Bildinformationen von Quelle und Ziel ermöglicht:

MODUS 0: immer 0
MODUS 1: Quelle AND Ziel
MODUS 2: Quelle And (NOT Ziel)
MODUS 3: Quelle
MODUS 4: (NOT Quelle) AND Ziel
MODUS 5: Ziel
MODUS 6: Quelle XOR Ziel
MODUS 7: Quelle OR Ziel
MODUS 8: NOT(Quelle OR Ziel)
MODUS 9: NOT(Quelle XOR Ziel)
MODUS 10: NOT Ziel
MODUS 11: Quelle OR (NOT Ziel)
MODUS 12: NOT Quelle

MODUS 13: (NOT Quelle) OR Ziel

MODUS 14: NOT(Quelle AND Ziel)

MODUS 15: immer 1

Wird die Angabe Modus weggelassen, so wird nach Modus 3 verfahren.

Bit-Blits in den Bildschirm berücksichtigen im Gegensatz zur GEM-VDI-Funktion ein gesetztes Grafikfenster (siehe CLIP).

Bei Bit-Blits innerhalb des Bildschirms gelten die jeweils kleineren Höhen und Breiten.

Der Speicherplatzbedarf für einen Bit-Blit in den Speicher läßt sich folgendermaßen berechnen:

6+(Breite+15) SHR 4 * Höhe * 2 * Farbebenen.

Die sechs Wort Vorspann setzen sich so zusammen:

1. Anzahl Bitplanes * 2; entspricht 1 SHL (3-Auflösung)
2. Breite in Pixel
3. Höhe in Pixel

Farbebenen:	s/w	1
	4 Farben	2
	16 Farben	4
	256 Farben	8

Beispiel:

```
0 Bildspeicher= MEMORY(6+65 SHR 4*(50+15)*2*4)
1 PELLIPSE 50,50,30,20
2 BITBLT 25,20,50,50 TO Bildspeicher
3 WAIT .5
4 CLS
5 BITBLT Bildspeicher TO 20,20,50,50
6 WAIT .5
7 BITBLT 20,20,50,50 TO 20,30,50,50,6
8 WAIT .5
9 PCIRCLE 45,55,8
```

BLOAD

Typ: Befehl

Syntax: BLOAD <Stringausdruck>[,<num.Ausdruck>]

BLOAD <Dateiname>[,<Startadresse>]

Erklärung: Die genannte Datei wird ab der Startadresse in den Speicher geladen. Fehlt dieser Ausdruck, so wird in den Bildschirm (logische Bildschirmadresse) geladen.

Beispiel:
```
Speicher= MEMORY(100)
BLOAD "BEISPIEL.DAT",Speicher
```

BOX

Typ: Befehl

Syntax: BOX <num.Ausdruck>,<num.Ausdruck>{ TO <num.Ausdruck>,<num.Ausdruck>|,<num.Ausdruck>,<num.Ausdruck>}

BOX <X>,<Y>{ TO <X2>,<Y2>|<Breite>,<Höhe>}

Erklärung: Zeichnet ein nicht gefülltes Rechteck auf den Bildschirm. Dabei sind entweder zwei gegenüberliegende Ecken anzugeben, oder eine Ecke, Breite und Höhe des Rechtecks.

Farbe, Linienstil und -breite können über LINE COLOR, LINE STYLE bzw. LINE WIDTH bestimmt werden.

Ist mittels CLIP ein Bildfenster definiert, wird außerhalb dieses Bereichs nicht gezeichnet.

Siehe auch RBOX, PBOX, PRBOX.

Beispiel:
```
0 BOX 20,30,70,40
```

BRK

Typ: Befehl

Syntax: BRK

Erklärung: Dieser Befehl dient ausschließlich zur Fehlersuche in compilierten Programmen. Er löst im Compilat eine Ausnahmebehandlung "Illegal Instruction" aus. Damit ist es möglich, ein Programm an einer bestimmten Stelle zu unterbrechen, um so die Fehlersuche mit einem Debugger zu beginnen.

Im Interpreter führt dieser Befehl zu einem "Internal Error". Dies ist jedoch normal und hat keine Bedeutung.

BSAVE

Typ: Befehl

Syntax: BSAVE <Stringausdruck>[,<num.Ausdruck>,<num.Ausdruck>]

BSAVE <Dateiname>[,<Startadresse>,<Länge>]

Erklärung: Speichert den ab Startadresse liegenden Speicherbereich unter Dateiname ab. Fehlen Adresse und Länge, wird der Bildschirm abgespeichert. Es werden direkt die Bilddaten ohne Farbpalette abgespeichert, d.h. je nach Bildschirmauflösung entsteht eine Datei mit 32000 Bytes (für ST-Auflösungen) oder 153600 Bytes (für TT-Auflösungen).

Beispiel:
```
BSAVE "BILDSCH.DAT"
```

CALL

Typ: Befehl

Syntax: CALL <num. Variable>[(][L]<num.Ausdruck>][[,[L]<num.Ausdruck>])]

Erklärung: Ein Maschienensprache-Unterprogramm wird aufgerufen. Die Startadresse ist durch die numerische Variable gegeben. Die folgenden Ausdrücke werden in umgekehrter Reihenfolge auf dem Stack übergeben d.h. der erste Parameter befindet sich zu unterst auf dem Stack also bei 4(SP). Den Ausdrücken darf ein "L " voran gestellt werden, um den betreffenden Parameter als LONG zu übergeben. Ansonsten wird immer ein WORD übergeben.

Das Maschinenprogramm muß mit RTS enden und wird im Supervisor-Modus aufgerufen. Es darf alle Register verändern außer USP und SSP. Im Register A0 erhält es die Adresse des Segmentpointers übergeben. Es ist damit z.B. möglich mit Hilfe übergebener Zeiger auf eine bestimmte Variable zuzugreifen. Das Maschinenprogramm kann im Register D0 einen Wert zurückgeben. Dieser befindet sich nach der Ausführung des Maschinenprogra... RESERVED(0). Fehler, wie "Adress Error" oder "Bus Error", die Ihr Maschinenprogramm v... ...sacht, werden in der Zeile des CALL-Befehls gemeldet.

Beispiel:
```
Ø Test$ = "PNJLSPO/TPGUXBSF"
1 CALL Decode ( L &Test$, 1 ) 'Zeiger auf Test$ und
                              "1" übergeben
2 PRINT Test$

OMIKRON.SOFTWARE
```

Das Maschinenprogramm:
```
move.l  4(SP),A1          ; Zeiger auf String holen
move.w  8(SP),D1          ; Parameter (die "1")
                          holen
```

```
        move.l  (A1),A2          ; Zeiger ins
                                 Stringsegment bilden
        adda.l  28(A0),A2
        move.w  4(A1),D0         ; Länge des Strings
        subq.w  #1,D0            ; wg. DBRA

loop:
        move.b  (A2),D2          ; ein Zeichen des Strings
                                 holen
        sub.b   D1,D2            ; Zeichen modifizieren
        move.b  D2,(A2)+         ; und Zeichen
                                 zurückschreiben
        dbra    D0,loop          ; für alle Zeichen des
                                 Strings
        rts
```

CASE

Typ: Befehl

Syntax: CASE <Ausdruck> [TO <Ausdruck>][[,<Ausdruck> [TO <Ausdruck>]]]

Erklärung: CASE leitet innerhalb einer SELECT-CASE Anweisung einen Teilzweig ein. Dieser Programmzweig wird dann ausgeführt, wenn eine der genannten Möglichkeit auf den bei SELECT genannten Ausdruck zutrifft. Mehrere Möglichkeiten werden einfach mit Kommata abgetrennt aufgezählt. Auch Bereiche sind mit TO möglich (siehe SELECT).

CDBL

Typ: Funktion

Syntax: CDBL(<num.Ausdruck>)

Erklärung: Wandelt einen beliebigen numerischen Ausdruck in doppelt-genaues Fließkomma-Format um.

Siehe auch CINT, CINTL, CSNG.

Beispiel:

```
0 A=1
1 B=3
2 PRINT CDBL(A/B)          'Genauigkeit: 9.5 Stellen
                           Nachkomma
```

```
3 PRINT CDBL(A)/CDBL(B)        'Genauigkeit:  17  Stellen
                                Nachkomma

 .333333333372138 44
 .3333333333333333
```

CHAIN

Typ: Befehl

Syntax: CHAIN [MERGE]<Stringausdruck>[,<Marke>]

CHAIN [MERGE]<Dateiname>[,<Start-Marke>]

Erklärung: Lädt die Datei als Basic-Programm und startet es. Durch MERGE wird das Programm zum aktuellen hinzugeladen. Bei gleichen Zeilennummern wird die alte Zeile gelöscht (siehe auch RENUM).

Das zu ladende Programm muß als ASCII-Datei vorliegen (als Block gesichert oder SAVE ,A).

Anders als beim Befehl RUN können Variablenwerte übergeben werden (siehe COMMON).

Zusätzlich kann eine Startmarke angegeben werden, an der das aufgerufene Programm starten soll. Fehlt diese Angabe, so wird am Programmanfang gestartet.

Beispiel:
```
CHAIN "BEISPIEL.BAS",23
```

CHDIR

Typ: Befehl

Syntax: CHDIR <Stringausdruck>

Erklärung: Legt das Standard-Laufwerk und den Standardpfad auf den in Stringausdruck genannten Pfad fest. Es ist möglich, einen kompletten Pfad zu benennen, oder durch folgende Stringausdrücke innerhalb des Pfades zu wandern:

"..": Ordner schließen, zurück auf nächsthöhere Ebene

"\" : zurück ins Hauptverzeichnis

"[\]<Ordner>[[\<Ordner>]]": Ordner (und Unterordner) öffnen, mit führendem \ vom Hauptverzeichnis aus

Beispiel:
```
CHDIR "A:\.BAS"
CHDIR "\"
CHDIR "B:\BASIC\PROGR"
```

CHR$

Typ: Funktion

Syntax: CHR$(<num.Ausdruck>)

Erklärung: Erzeugt einen ein Zeichen langen String entsprechend der ASCII-Tabelle. CHR$ ist die Umkehrfunktion zu ASC.

Siehe auch ASCII-Tabelle.

Beispiel:

```
0 PRINT CHR$(7)
1 PRINT CHR$(80); CHR$(76)
2 A$= CHR$(189)+" by OMIKRON."
3 PRINT A$

PL
© by OMIKRON.
```

CINT

Typ: Funktion

Syntax: CINT(<num.Ausdruck>)

Erklärung: Wandelt einen beliebigen numerischen Wert in Integer-Wort-Format (ganze Zahlen von -32768 bis +32767) um. Gerundet wird wie bei INT. Der Definitionsbereich der Funktion umfaßt natürlich gerade den Integer-Wort-Bereich. Andere Werte führen zu "OVERFLOW".

Siehe auch CDBL, CINTL, CSNG.

Beispiel:

```
0 A=1
1 B=3
2 PRINT CINT(A/B)
3 PRINT CINT(A)/CINT(B)
4 PRINT CINT(40000)

 0
 .33333333
? Overflow in 4
```

CINTL

Typ: Funktion

Syntax: CINTL(<num.Ausdruck>)

Erklärung: Wandelt einen beliebigen numerischen Wert in Integer-Langwort-Format (ganze Zahlen von -214783658 bis +214783657) um. Gerundet wird wie bei INT. Der Definitionsbereich der Funktion umfaßt den Integer-Langwort-Bereich. Andere Werte führen zu "OVERFLOW".

Siehe auch CDBL, CINT, CSNG.

Beispiel:

```
0 A=1
1 B=3
2 PRINT CINTL(A/B)
3 PRINT CINTL(A)/CINTL(B)
4 PRINT CINTL(1E+10)

 0
 .33333333
? Overflow in 4
```

CIRCLE

Typ: Befehl

Syntax: CIRCLE <num.Ausdruck>,<num.Ausdruck>,<num.Ausdruck>[,<num.Ausdruck>,<num.Ausdruck>]

CIRCLE <X>,<Y>,<Radius>[,<Startwinkel>,<Endwinkel>]

Erklärung: Zeichnet einen nicht gefüllten Kreis auf den Bildschirm. <X> und <Y> geben den Kreismittelpunkt an. Optional können Start- und Endwinkel in Zehntel-Graden angegeben werden. Hierbei ist Winkel=0 rechts vom Mittelpunkt, Winkel=900 oberhalb des Mittelpunkts etc.

Farbe, Linienstil und -breite können über LINE COLOR, LINE STYLE bzw. LINE WIDTH bestimmt werden.

Ist mittels CLIP ein Bildfenster definiert, wird außerhalb dieses Bereichs nicht gezeichnet.

Siehe auch ELLIPSE, PCIRCLE, PELLIPSE.

Beispiel:

```
0 CIRCLE 70,50,40
1 CIRCLE 70,50,30,0,1800
2 CIRCLE 60,50,20,1800,3200
3 CIRCLE 70,50,25,500,600
4 CIRCLE 70,50,25,1200,1300
```

CLEAR

Typ: Befehl

Syntax: CLEAR [<num.Ausdruck>][,<num.Ausdruck>]

CLEAR [<GEMDOS-Speicher>][,<Stack-Größe>]

Erklärung: Löscht alle Variableninhalte und Dimensionierungen. Offene Dateien werden geschlossen, ON ERROR GOTO- und ON TIMER GOSUB-Aufrufe werden abgeschaltet. COMMON-Anweisungen werden gelöscht. Optional können die Größe des GEMDOS-Speichers und/ oder des Prozessor-Stapels bestimmt werden. Letztere legen Sie mit Stack-Größe fest.

Standardmäßig sind für diese 65536, bzw. 4096 Bytes belegt.

Der GEMDOS-Speicher wird zum Reservieren vom Speicher mit MEMORY und von bestimmten Betriebssystemfunktionen benötigt. Wenn Sie also selbst eine größere Menge an GEMDOS-Speicher anfordern, so müssen Sie ihm entsprechend groß einstellen.

Stapelspeicherplatz wird vom Basicprogramm beim Aufrufen von Unterprogrammen, von lokalen Variablen und zum Sortieren benötigt. Programme mit rekursiven Unterprogrammaufrufen benötigen deshalb unter Umständen mehr Stapelplatz.

Zu beachten ist: Der Verbrauch an Stapelplatz ist im Interpreter und im Compilaten unterschiedlich und muß gegebenfalls getrennt ermittel werden.

Siehe auch FRE und MEMORY.

Beispiel:

```
a) CLEAR 32768,16384
CLEAR ,4096

b) ' 50 KB BASIC-Speicher:
   CLEAR 0
   CLEAR FRE(0)-50000
```

CLIP

Typ: Befehl

Syntax: CLIP [<num.Ausdruck>,<num.Ausdruck>{ TO <num.Ausdruck>,<num.Ausdruck>|,<num.Ausdruck>,<num.Ausdruck>}]

CLIP [<X>,<Y>{ TO <X2>,<Y2>|,Breite,Höhe}]

Erklärung: Definiert ein Grafik-Fenster, das den Zeichenbereich für folgende Zeichenbefehle begrenzt: BITBLT, BOX, CIRCLE, DRAW, ELLIPSE, FILL, PBOX, PCIRCLE, PELLIPSE, POLYGON, PPOLYGON, PRBOX, RBOX, TEXT. Außerhalb dieses Fenster wird die Zeichenoperation zwar ausgeführt, der Bildschirm wird dabei jedoch nicht verändert.

Das Fenster kann entweder durch Angabe zweier gegenüberliegender Eckpunkte oder eines Eckpunktes mit Höhe und Breite definiert werden.

Ohne Parameterangaben wird das Grafik-Fenster ausgeschaltet (dies ist auch die Standard-Einstellung). Die Zeichenbefehle werden nun in keiner Weise begrenzt. Es ist deshalb möglich, daß z.B. durch falsch berechnete Koordinaten, ungewollt wichtiger Speicher überschrieben wird (Abstürze und Datenverluste möglich). Sie sollten daher immer wenn Sie mit Grafikbefehlen arbeiten das Grafik-Fenster auf den gesamten Bildschirm einstellen: "CLIP 0,0,W_PIXEL,H_PIXEL".

Beispiel:

```
0 CLIP 30,30,20,20
1 PCIRCLE 30,30,20
2 CLIP 60,60 TO 150,100
3 PRBOX 65,65,200,200
```

CLOSE

Typ: Befehl

Syntax: CLOSE [<num.Ausdruck>[[,<num.Ausdruck>]]]

CLOSE [<Dateinummer>[[,<Dateinummer>]]]

Erklärung: Schließt die mit der Dateinummer assozierte(n) Datei(en). Ohne Angabe von Parametern werden alle Dateien geschlossen.

Siehe auch OPEN.

Beispiel:

```
CLOSE
CLOSE 1,7,0m
```

CLS

Typ: Befehl

Syntax: CLS

Erklärung: Der gesamte Bildschirm wird gelöscht und der Cursor springt in die linke obere Ecke des Text-Fensters.

Beispiel:

```
0 PELLIPSE 50,50,40,20
1 WAIT .5
2 CLS
```

CMD

Typ: Funktion

Syntax: CMD <num.Ausdruck>

CMD <Dateinummer>

Erklärung: Lenkt die Ausgaben von PRINT, LIST etc. auf eine Datei um. Die der Dateinummer entsprechende Datei muß zuvor mittels OPEN geöffnet worden sein. Durch die Möglichkeit Dateien auch auf den Drucker zu öffnen (siehe OPEN), kann die Ausgabe auch auf den Drucker oder die serielle Schnittstelle umgelenkt werden.

Ist der numerische Ausdruck null, so kommen die Ausgaben wieder zum Bildschirm. Die Ausgabe folgender Befehle wird durch CMD umgelenkt: PRINT, PRINT#, LPRINT, WRITE, LIST, LLIST, DUMP, LDUMP. Bei CMD sind LLIST und LDUMP vor LIST und DUMP vorzuziehen, da LLIST im Gegensatz zu LIST keine Steuerzeichen ausgibt, was beim Drucken oder Schreiben in eine Datei hinderlich wäre.

Beispiel:
```
OPEN "P",3: CMD 3
CMD 0
```

COLOR

Erklärung: siehe BITBLT, LINE COLOR, FILL COLOR

COMMON

Typ: Befehl

Syntax: COMMON <Variable>[[,Variable]]

Erklärung: Gibt die Variablen an, deren Inhalte bei Aufruf eines Programms mit CHAIN an dieses übergeben werden.

Der Befehl muß mit der gleichen Variablenliste sowohl im aufrufenden, als auch im aufgerufenen Programm am Anfang stehen, bzw. in der bei CHAIN genannten Start-Zeile.

Die Variableninhalte werden der Reihe nach zugewiesen. Eine Typenumrechnung (beispielsweise von Integer nach Fließkomma) wird automatisch durchgeführt. Einer String-Variablen eine numerische Variable zuzuweisen, oder umgekehrt, ist unzulässig. Felder können nicht als ganzes übergeben werden. Die Zuweisung einzelner Elemente ist aber möglich.

Ein COMMON wird durch CLEAR gelöscht.

Beispiel:
```
COMMON A$,De,N(23),N(24),N(25)
```

COMPILER

Typ: Befehl

Syntax: COMPILER {OFF | ON | <Stringausdruck>}

Erklärung: Zwischen COMPILER OFF und COMPILER ON stehende Programmteile werden beim Compilieren nicht mitübersetzt und fallen so aus dem Programm.

Im Stringausdruck können dem Compiler Steuerworte (z.B. "Multitasking always" etc.) übergeben werden, die vom Interpreter nicht beachtet werden.

Nähere Informationen hierzu können Sie dem Compiler-Handbuch entnehmen

Beispiel:
```
COMPILER OFF
COMPILER "Trace_On"
```

COMPILER

Typ: Funktion

Syntax: COMPILER

Erklärung: Ergibt den Wert -1, wenn das laufende Programm als Compilat, und 0, wenn es im Interpreter läuft.

Beispiel:
```
Ø IF NOT(COMPILER) THEN PRINT "Interpreter!"
1 IF COMPILER THEN PRINT "Compiler!"

Interpreter!
```

CONT

Typ: Befehl

Syntax: CONT [<Marke>][TO <Marke>]

CONT [<Start-Marke>][TO <End-Marke>]

Erklärung: Führt die Programmausführung nach Abbruch durch CONT OL-C oder STOP fort, an der Unterbrechungsstelle fort, bzw. an der Start-Marke. Optional kann die Programmausführung an der End-Marke automatisch wieder angehalten werden.

Der Befehl CONT kann nicht mehr ausgeführt werden, nachdem am Programm Änderungen vorgenommen wurden, 16 Fehlermeldungen ausgegeben wurden oder nach einer Fehlermeldung "? Out of Memory".

Beispiel:
```
CONT Berechnung TO Ergebnis
' (Der Programmteil zwischen Marke -Berechnung und -
Ergebnis wird durchlaufen, danach erneut gestoppt)
```

CONTINUE

Typ: Befehl

Syntax: CONTINUE

Erklärung: CONTINUE innerhalb einer SELECT-CASE-Anweisung übergeht jeweils die nächste CASE-Bedingung und führt das Programm hinter dem CASE weiter aus.

Beispiel:
```
Ø A$="Q"
1 SELECT A$
2   CASE "Q"
3     PRINT "Das Zeichen war ein Q"
4     CONTINUE
5   CASE "Ø" TO "9"
6     PRINT "Das Zeichen war eine Ziffer"
7 END_SELECT

Das Zeichen war ein Q
Das Zeichen war eine Ziffer
```

COPY

Typ: Befehl

Syntax: COPY <Dateiname> TO <Dateiname>

COPY <Quell-Dateiname> TO <Ziel-Dateiname>

Erklärung: Erstellt von Quell-Datei eine Kopie, mit dem Namen Ziel-Datei. Quell- und Ziel-Datei müssen jeweils mit Pfad angegeben werden. Fehlt in Ziel-Datei der Dateiname, so erhält die Kopie den Namen des Originals.

Kann die Datei nicht gefunden werden, wird der Befehl abgebrochen und ohne Fehlermeldung im Programmablauf fortgefahren.

Beispiel:
```
COPY "A:\*.BAS" TO "C:\"
```

COS

Typ: Funktion

Syntax: COS(<num.Ausdruck>)

Erklärung: Berechnet den Cosinus des numerischen Ausdrucks. Das Ergebnis ist abhängig vom eingestellten Winkelmodus (siehe DEG, RAD).

Beispiel:

```
0 PRINT COS( PI )
1 DEG
2 PRINT COS(90)

-1
 0
```

COSEC

Typ: Funktion

Syntax: COSEC(<num.Ausdruck>)

Erklärung: Berechnet den Cosecans des numerischen Ausdrucks. Das Ergebnis ist abhängig vom eingestellten Winkelmodus (siehe DEG, RAD).

Beispiel:

```
0 PRINT COSEC( PI /2)
1 DEG
2 PRINT COSEC(180)

 1
-6.8356528E+8
```

COSECH

Typ: Funktion

Syntax: COSECH(<num.Ausdruck>)

Erklärung: Berechnet den Cosecans Hyperbolicus des numerischen Ausdrucks.

Beispiel:

```
0 PRINT COSECH(1)

 .85091813
```

COSH

Typ: Funktion

Syntax: COSH(<num.Ausdruck>)

Erklärung: Berechnet den Cosinus Hyperbolicus des numerischen Ausdrucks.

Beispiel:

```
Ø PRINT COSH(.5)

 1.127626
```

COT

Typ: Funktion

Syntax: COT(<num.Ausdruck>)

Erklärung: Berechnet den Cotangens des numerischen Ausdrucks. Das Ergebnis ist abhängig vom eingestellten Winkelmodus (siehe DEG, RAD).

Beispiel:

```
Ø PRINT COT(3)
1 DEG
2 PRINT COT(90)

-7.0152526
 Ø
```

COTH

Typ: Funktion

Syntax: COTH(<num.Ausdruck>)

Erklärung: Berechnet den Cotangens Hyperbolicus des numerischen Ausdrucks. Der numerische Wert muß ungleich null sein.

Beispiel:

```
Ø PRINT COTH(1)

 1.3130353
```

CSNG

Typ: Funktion

Syntax: CSNG(<num.Ausdruck>)

Erklärung: Wandelt einen beliebigen numerischen Ausdruck in einfach-genaues Fließkomma-Format um.

Siehe auch CDBL, CINT, CINTL.

Beispiel:

```
Ø A#=1#
1 B#=3#
2 PRINT CSNG(A#/B#)
3 PRINT CSNG(A#)/CDBL(B#)

 .33333333
 .33333333333333333
```

CSRLIN

Typ: Funktion

Syntax: CSRLIN

Erklärung: Ergibt die Zeile, in der sich der Cursor gerade befindet. Die oberste Zeile hat die Nummer eins, im Gegensatz zu @, das die oberste Zeile mit null zählt.

Siehe auch POS, LOCATE, @.

Beispiel:

```
Ø PRINT ·(10,Ø);"Hier ist Zeile: "; CSRLIN

Hier ist Zeile: 11
```

CVD

Funktion

Syntax: CVD(<Stringausdruck>)

Erklärung: Die ersten 10 Zeichen des Stringausdrucks werden in eine doppelt-genaue-Fließkommazahl umgewandelt. Umkehrfunktion zu MKD$.

Die Funktion dient in erster Linie zum Lesen von Binär-Dateien, welche Fließkommazahlen nicht in ihrer lesbaren ASCII-

Darstellung, sondern in ihrer binär Darstellung enthalten.

Beispiel:

```
PRINT CVD("1234567890")
PRINT CVD( MKD$(1234567890))

 Ø.7708077718716055D+1895
 1234567890
```

CVI

Typ: Funktion

Syntax: CVI(<Stringausdruck>)

Erklärung: Die ersten beiden Zeichen des Stringausdrucks werden in eine Integer-Wort-Zahl umgewandelt. Umkehrfunktion zu MKI$.

Die Funktion dient in erster Linie zum Lesen von Binär-Dateien, welche Integer-Wort-Zahlen nicht in ihrer lesbaren ASCII-Darstellung, sondern in ihrer binär Darstellung enthalten.

Beispiel:

```
PRINT CVI("12")
PRINT CVI( MKI$(12))

 12594
 12
```

CVIL

Typ: Funktion

Syntax: CVIL(<Stringausdruck>)

Erklärung: Die ersten 4 Zeichen des Stringausdrucks werden in eine Integer-Langwort-Zahl umgewandelt. Umkehrfunktion zu MKIL$.

Die Funktion dient in erster Linie zum Lesen von Binär-Dateien, welche Integer-Lang-Wortzahlen nicht in ihrer lesbaren ASCII-Darstellung, sondern in ihrer binär Darstellung enthalten.

Beispiel:

a)
```
PRINT CVIL("1234")
PRINT CVIL( MKIL$(1234))

 825373492
 1234
```

b)
```
0 REPEAT A$=INKEY$ UNTIL LEN(INKEY$)
1 PRINT CVIL(A$)
```

CVS

Typ: Funktion

Syntax: CVS(<Stringausdruck>)

Erklärung: Die ersten 6 Zeichen des Stringausdrucks werden in eine einfach-genaue-Fließkommazahl umgewandelt. Umkehrfunktion zu MKS$.

Die Funktion dient in erster Linie zum Lesen von Binär-Dateien, welche einfach-genaue-Fließkommazahlen nicht in ihrer lesbaren ASCII-Darstellung, sondern in ihrer binär Darstellung enthalten.

Beispiel:
```
PRINT CVS("123456")
PRINT CVS( MKIL$(123456))

 7.7080777E+1894
 123456
```

DATA

Typ: Befehl

Syntax: DATA <Ausdruck>[[,<Ausdruck>]]

Erklärung: Die numerischen und/oder String-Ausdrücke sind der Reihe nach als "offene Zuweisung" im Programm gespeichert. Offene Zuweisung bedeutet, daß ein numerischer Ausdruck oder ein String nicht direkt einer Variablen zugewiesen wird, sondern ersteinmal in einer DATA-Liste gespeichert wird. Erst im weiteren Programmverlauf findet die Zuweisung nach Auslesen mittels READ statt. Die DATA-Zuweisung hat auch dann Gültigkeit, wenn sie vom Programm noch nicht durchlaufen wurde. Sie kann also an Stellen des Programms stehen, die nie durchlaufen werden. Trotzdem können die Daten mittels READ problemlos "gelesen" werden.

Mit dem Befehl RESTORE können Sie den Zeiger verschieben, der auf die nächste offene Zuweisung zeigt (siehe RESTORE).

Beispiel:
```
0 DATA "Hund",A$
1 Inh=2
2 READ A$,B$,C$,A
3 PRINT A$,B$,C$,A
4 DATA "Katze",23*Inh

Hund     Hund     Katze     46
```

DATE$

Typ: Befehl

Syntax: DATE$=<Stringausdruck>

Erklärung: Weist der Systemuhr das Datum Stringausdruck zu.
Stringausdruck ist in Abhängigkeit vom Ländermodus anzugeben:

```
MODE "D":   "TT.MM.JJ"
MODE "GB":  "TT/MM/JJ
MODE "USA": "MM/TT/JJ"
(T=Tag, M=Monat, J=Jahr)
```

Beispiel:

```
0 MODE "D"
1 DATE$="21.08.83"
2 PRINT DATE$

21.08.83
```

DATE$

Typ: Funktion

Syntax: DATE$

Erklärung: Gibt einen Stringausdruck des Datums entspechend dem eingestellten Ländermodus:

```
MODE "D":   "TT.MM.JJ"
MODE "USA": "MM/TT/JJ"
(T=Tag, M=Monat, J=Jahr)
```

Beispiel:

```
0 MODE "D"
1 DATE$="21.08.83"
2 MODE "USA"
3 PRINT DATE$

08/21/83
```

DEF FN

Typ: Befehl

Syntax: DEF FN <Funktionsname>[([<Parameterliste>])][=<Ausdruck>]

Erklärung: Definiert eine Funktion. Es können ein oder mehrere Parameter festgelegt werden, die beim Aufruf angegeben werden müssen. Die Parameter sind automatisch lokale Variablen (siehe auch LOCAL).

Der Wert der Funktion kann direkt durch =Ausdruck angegeben werden (in diesem Fall ist die Funktionsdefinition einzeilig) oder alternativ nach einer ein- oder mehrzeiligen Berechnung durch RETURN <Ausdruck> zurückgegeben werden. Der Funktionsname beinhaltet einen Variablentyp. Die Rückgabevariable sollte vom gleichen Typ sein.

Die Funktion ist im Programm über

N<Funktions-Variable>[(<Parameter>[[,Parameter]])] aufrufbar.

ie Funktionsdefinition kann an beliebiger Stelle im Programm erscheinen. Eine mehrzeilige Funktionsdefinition darf während des Programmablauf nicht durchlaufen werden. Eine mehrzeilige Funktionsdefinition sollte mit END_FN beendet werden. Diese Anweisung wirkt wie ein normales RETURN und dient in erster Linie der besseren Strukturierung des Programms.

Beispiel:

```
0 PRINT FN Doppelt(13)
1 PRINT FN Primzahl$(13)
2 END
3 DEF FN Doppelt(A)=2*A    'einzeilige Funktion
4 DEF FN Primzahl$(A)      'mehrzeilige Funktion
5   LOCAL N,P$="Primzahl"
6   FOR N=2 TO SQR(A)
7     IF A MOD N=0 THEN P$="keine Primzahl"
8   NEXT N
9   RETURN P$
0 END_FN

26
Primzahl
```

DEF PROC

Typ: Befehl

Syntax: DEF PROC <Prozedurname>[([R]<Variable>[[,[R]<Variable>]])]

Erklärung: Leitet eine Prozedurdefinition ein. Es können ein oder mehrere Parameter definiert werden. Die Parameter sind lokale Variablen (siehe auch LOCAL). Durch vorangestelltes R werden die Werte nach Beendigung der Prozedur an die im Prozeduraufruf genannten Variablen übergeben.

Die Prozedur wird durch ein RETURN oder besser durch ein END_PROC beendet.

Zum Aufruf ist der Prozedurnamen zu nennen und gegebenenfalls die in Klammern gesetzten und durch Komma getrennten Ausrücke. Ist in der Definition an einer Stelle ein R vor die Variable gestellt, so muß an entsprechender Stelle im Aufruf eine Variable stehen. An diese wird am Ende der Prozedur der Inhalt der Prozedur-Variablen übergeben.

Die Prozedurdefinition kann an beliebiger Stelle im Programm erscheinen. Die Prozedurdefinition darf während des Programmablaufs nicht durchlaufen werden.

Beispiel:

```
 1 Zentriere "Dieser Text erscheint zentriert"
 1 Gesperrt "OMIKRON.SOFTWARE",2
 2 END
 3 DEF PROC Zentriere(Text$)
 4   PRINT TAB((W_CHAR-LEN(Text$))/2);Text$;
 5 END_PROC
 6 DEF PROC Gesperrt(Text$,Anzahl_Leerzeichen)
 7   LOCAL I
 8   FOR I=1 TO LEN(Text$)
 9     PRINT MID$(Text$,I,1);
10     PRINT " "*Anzahl_Leerzeichen;
11   NEXT I
12 END_PROC
```

```
        Dieser Text erscheint zentriert
O M I K R O N . S O F T W A R E
```

DEF USR

Typ: Befehl

Syntax: DEF USR=<num.Ausdruck>

Erklärung: Legt die Adresse fest, die beim Aufruf eines Maschinensprache-Programms durch USR angespungen wird (siehe auch USR).

Beispiel: `DEF USR Scroll_Screen`

DEFDBL

Typ: Befehl

Syntax: DEFDBL {Buchstabe|Buchstabe-Buchstabe}[[,{Buchstabe|Buchstabe-Buchstabe}]]

Erklärung: Es können einzelne Buchstaben oder Buchstabenbereiche (nach Alphabet) genannt werden. Die Buchstaben müssen groß geschrieben werden. Nachfolgend im Programmtext erscheinende Variablen ohne Postfix, die mit einem solchen Buchstaben beginnen werden automatisch als doppelt-genaue-Fließkomma-Variablen interpretiert.

Das nachträgliche Einfügen des Befehls verändert schon bestehende Variablenstrukturen nicht. Er ändert lediglich das Anzeigeverhalten des Editors.

Beispiel:
```
DEFDBL "A-Z" ' alle Variblen sind doppelt-genau
```

DEFER

Typ: ***Reserviert für zukünfige Anwendungen!***

Dieses Wort bitte nicht verwenden!!

DEFINT

Typ: Befehl

Syntax: DEFINT {Buchstabe|Buchstabe-Buchstabe}[[,{Buchstabe|Buchstabe-Buchstabe}]]

Erklärung: Es können einzelne Buchstaben oder Buchstabenbereiche (nach Alphabet) genannt werden. Nachfolgend im Programmtext erscheinende Variablen ohne Postfix, die mit einem solchen Buchstaben beginnen werden automatisch als Integer-Wort-Variablen interpretiert.

Das nachträgliche Einfügen des Befehls verändert schon bestehende Variablenstrukturen nicht. Er ändert lediglich das Anzeigeverhalten des Editors.

Beispiel:
```
DEFINT "I,J,K"
```

DEFINTL

Typ: Befehl

Syntax: DEFINTL {Buchstabe|Buchstabe-Buchstabe}[[,{Buchstabe|Buchstabe-Buchstabe}]]

Erklärung: Es können einzelne Buchstaben oder Buchstabenbereiche (nach Alphabet) genannt werden. Nachfolgend im Programmtext erscheinende Variablen ohne Postfix, die mit einem solchen Buchstaben beginnen werden automatisch als Integer-Langwort-Variablen interpretiert.

Das nachträgliche Einfügen des Befehls verändert schon bestehende Variablenstrukturen nicht. Er ändert lediglich das Anzeigeverhalten des Editors.

Wichtig: Der Befehl wirkt 1. direkt bei der Eingabe und 2. beim Ausführen!

Beispiel: DEFINTL I,J,K,X-Z

DEFSNG

Typ: Befehl

Syntax: DEFSNG {Buchstabe|Buchstabe-Buchstabe}[[,{Buchstabe|Buchstabe-Buchstabe}]]

Erklärung: Es können einzellne Buchstaben oder Buchstabenbereiche (nach Alphabet) genannt werden. Die Buchstaben müssen groß geschrieben werden. Nachfolgend im Programmtext erscheinende Variablen ohne Postfix, die mit einem solchen Buchstaben beginnen werden automatisch als einfach-genaue-Fließkomma-Variablen interpretiert.

Das nachträgliche Einfügen des Befehls verändert schon bestehende Variablenstrukturen nicht. Er ändert lediglich das Anzeigeverhalten des Editors.

Beispiel: DEFSNG "A-Z"

DEFSTR

Typ: Befehl

Syntax: DEFSTR {Buchstabe|Buchstabe-Buchstabe}[[,{Buchstabe|Buchstabe-Buchstabe}]]

Erklärung: Es können einzellne Buchstaben oder Buchstabenbereiche (nach Alphabet) genannt werden. Die Buchstaben müssen groß geschrieben werden. Nachfolgend im Programmtext erscheinende Variablen ohne Postfix, die mit einem solchen Buchstaben beginnen werden automatisch als String-Variablen interpretiert.

Das nachträgliche Einfügen des Befehls verändert schon bestehende Variablenstrukturen nicht. Er ändert lediglich das Anzeigeverhalten des Editors.

Beispiel: DEFSTR "T"

DEG

Typ: Befehl

Syntax: DEG

Erklärung: Schaltet für trigonometrische Funktionen auf Berechnung nach Gradmaß um (0 bis 360). Standardeinstellung ist Bogenmaß (0 bis 2), siehe RAD.

WICHTIG: Wenn der Befehl zusammen mit dem COMPILER 4.0 verwendet wird und die Compilate den Coprozessor auf ATARI TT ausnutzen, ist folgendes zu beachten: Die Befehle DEG und RAD wirken als Steuerworte d.h. der Rechenmodus wird ab der Stelle, ab der der Befehl im Programm steht umgestellt. Wann der Befehl ausgeführt wird, ist gleichgültig. Falls das Programm wie im Beispiel einen unverzweigten Ablauf von oben nach unten hat, ist die unterschiedliche Behandlung belanglos. Es ist bei Coprozessor-Compilaten jedoch nicht möglich, den Winkelmodus in Abhängigkeit von einer Bedingung zu setzen. Hier muß man selbst Vorkehrung für eine Umrechnung vorsehen.

Beispiel:

```
DEG: PRINT SIN(45)
RAD: PRINT SIN(45)

 .70710678
 .85090352
```

DET

Typ: Funktion

Syntax: DET(<Matrixname>([<num.Ausdruck>],[<num.Ausdruck>))

Erklärung: Diese Funktion berechnet die Determinante einer quadratischen Matrix. Die Matrix muß in Form eines zweidimensionalen Fließkommafeldes vorliegen. Die Größe der Matrix muß für beide Dimensionen angegeben werden. Wenn die Größenangabe fehlt, werden automatisch die bei [illegible]Dimensonieren verwendeten Maximalwerte verwendet.

```
DATA 1.,2.,3.,4.,5.,6.,7.,8.,9.
1 DIM Matrix!(2,2)
2 FOR I=0 TO 2
3   FOR J=0 TO 2
4     READ Matrix!(I,J)
5   NEXT J
6 NEXT I
7 PRINT DET(Matrix!(,))

0
```

DIM

Typ: Befehl

Syntax: DIM <Feldname>(<num.Ausdruck>[[,<num. Ausdruck]])[[,<Variable>(<num.Ausdruck>[[,<num. Ausdruck]])]]

Erklärung: Definiert eine oder mehrere Variablen- Felder und gibt durch die numerischen Ausdrücke deren maximale Feldgröße an.

Eindimensionale Felder (nur ein numerischer Ausdruck) können beliebig groß sein. Für Mehrdimensionalige Felder gilt: erste Dimension kleiner 65536, alle weiteren durchmultipliziert kleiner 65536.

Die Benutzung von Feldvariablen ist erst nach einer DIM-Anweisung möglich, Ausnahme: Eindimensionale Felder bis elf Elemente (0 bis 10) müssen im Interpreter nicht dimensioniert werden. Allein aus Gründen der Kompatibilität zum Compiler sollten jedoch stets alle Feldvariablen dimensioniert werden.

Nach einer Definition können Feldgrößen nur noch mit folgenden Einschränkungen verändert werden:

Wird eine andere als die letzte Dimension eines Feldes verändert, verschieben sich die Feldinhalte. Wird das Feld um eine weitere Dimension erweitert, gehen die Feldinhalte verloren.

Die Variablentypen Flag (Postfix %F) und Byte (Postfix %B) können nur als Feld verwendet werden.

Beispiel:
```
DIM A%(10),Flags%F(100),Wert#(50,50)
```

DMA_SOUND

Typ: **Reserviert für zukünftige Anwendungen.**

Dieses Wort bitte nicht verwenden!!

DRAW

Typ: Befehl

Syntax: DRAW [TO] <num.Ausdruck>,<num.Ausdruck>[[TO <num. Ausdruck>,<num.Ausdruck>]]

DRAW [TO] <X>,<Y>[[TO <Xn>,<Yn>]]

Erklärung: Zeichnet an der durch X und Y gegebenen Koordinate einen Punkt. Durch Angabe von TO vor dieser Koordinate wird eine Linie vom letzten gezeichneten Punkt aus gezeichnet.

Optional können weitere durch TO abgetrennte Koordinaten angegeben werden, die dann durch Linien verbunden werden.

Farbe, Linienstil und Linienbreite können über LINE COLOR, LINE STYLE, bzw. LINE WIDTH bestimmt werden.

Ist mittels CLIP ein Bildfenster definiert, wird außerhalb dieses Bereiches nicht gezeichnet.

Siehe auch POLYGON und PPOLYGON.

Beispiel:

```
0 DRAW 100,100
1 DRAW 150,100
2 DRAW TO 100,150
3 DRAW 70,70 TO 20,30 TO 60,40
```

DUMP

Typ: Befehl

Syntax: DUMP [{Buchstabe|Buchstabe-Buchstabe}[[,{Buchstabe|Buchstabe-Buchstabe}]]]

Erklärung: Alle Variablen werden mit Inhalt auf dem Bildschirm ausgegeben. Bei Feldern wird jedoch nur der Dimensionsbereich angegeben.

Optional können einzelne Buchstaben oder Buchstabenbereiche (nach Alphabet) genannt werden. Dann werden nur solche Variablen aufgelistet, die mit einem der genannten Buchstaben beginnen.

Beispiel:

```
0 Cop$="OMIKRON.Software":Betrag#=229.45#
1 DUMP B,C

COP$="OMIKRON.Software"
BETRAG#= 229.45
```

EDIT

Typ: Befehl

Syntax: EDIT [<num.Ausdruck>]

Erklärung: Springt vom Direkt-Modus in den Editor. Mit dem numerischen Ausdruck kann die Zeilennummer angegeben werden, an der der Cursor positioniert wird. Wird 0 oder -1 angegeben, so steht der Cursor in der ersten, bzw. letzten Programmzeile.

Der Editor kehrt mit Anklicken des Eintrags DIRECT MODE im FILE-Menü in den Direkt-Modus zurück.

Wichtige Anwendung: Mit EDIT ERL gelangt man direkt in die fehlerhafte Programmzeile

ELLIPSE

Typ: Befehl

Syntax: ELLIPSE <num.Ausdruck>,<num.Ausdruck>,<num.Ausdruck>,<num.Ausdruck>[,<num.Ausdruck>,<num.Ausdruck>]

ELLIPSE <X>,<Y>,<X-Radius>,<Y-Radius>[,<Startwinkel>,<Endwinkel>]

Erklärung: Zeichnet um den Mittelpunkt X,Y eine Ellipse mit den angegebenen Radien. Optional können Start- und Endwinkel in Zehntel-Graden angegeben werden. Hierbei ist Winkel=0 rechts vom Mittelpunkt, Winkel=900 oberhalb des Mittelpunkts etc.

Farbe, Linienstil und Linienbreite können über LINE COLOR, LINE STYLE, bzw. LINE WIDTH bestimmt werden.

Ist mittels CLIP ein Bildfenster definiert, wird außerhalb dieses Bereiches nicht gezeichnet.

Siehe auch CIRCLE, PELLIPSE, PCIRCLE.

ELSE

Typ: Befehl

Syntax: ELSE

Erklärung: Steht nach der IF-Bedingung und vor weiteren IF-Bedingungen, weiteren ELSE's oder ENDIF. Was nach ELSE steht, wird genau dann ausgeführt, wenn die letzte IF- Bedingung NICHT erfüllt, also FALSE ist. Siehe auch IF.

Beispiel:

```
Ø IF Ø=1 THEN
1   PRINT "Ø ist gleich 1"
2   PRINT "Das Un-fragt Hamlets Frage nach dem Sein!"
3 ELSE
4   PRINT "Ø ist ungleich 1"
5   PRINT "Wir haben nochmal Glück gehabt!"
6 ENDIF
```

END

Typ: Befehl

Syntax: END

Erklärung: Beendet den Programmablauf und schließt alle Dateien.

ENDIF

Typ: Befehl

Syntax: ENDIF

Erklärung: ENDIF beendet den abhängigen Programmzweig einer mehrzeiligen IF-Anweisung.(siehe IF)

Beispiel:

```
IF Count›Maximum THEN
   Count=Ø
   Max_Count+=1
ENDIF
```

END_FN

Typ: Befehl

Syntax: END_FN

Erklärung: Schließt eine Funktionsdefinition ab und sollte daher am Ende jeder Funktionsdefinition stehen. Mit END_FN kann kein Funktionswert zurückgeliefert werden. Rückgabewerte müssen mit dem Befehl RETURN realisiert werden. Im Unterschied zu END_FN kann RETURN innerhalb einer Funktion mehrmals verwendet werden (z.B. IF X THEN RETURN 5 ELSE RETURN 3). Allerdings können mit [CONTROL][D] nur Funktionen eingeklappt werden, die mit END_FN abgeschlossen sind.

Wenn Sie auf die pratische Möglichkeit des Einklappens verzichten möchten, können Sie Ihre Funktion auch mit RETURN abschließen.

END_PROC

Typ: Befehl

Syntax: END_PROC

Erklärung: Beendet eine Prozedur und entspricht etwa dem Befehl RETURN, darf jedoch nur genau einmal (und zwar am Ende der Prozedur) stehen. Nur eine mit END_PROC abgeschlossene Prozedur kann mit [CONTROL][D] eingeklappt werden.

EOF

Typ: Funktion

Syntax: EOF(<num.Ausdruck>) oder EOF(<Dateinummer>)

Erklärung: Ergibt -1 (=wahr), wenn das Ende der Datei erreicht ist, sonst 0. Die Datei muß zuvor mit OPEN geöffnet worden sein.

Achtung: Bei sequentiellen Dateien muß EOF ***VOR*** dem Lesen abgefragt werden, bei relativen Dateien ***NACH*** dem GET!

Beispiel:

```
0 OPEN "I",1,"C:\DESKTOP.INF"
1 WHILE NOT EOF(1)
2   In$=INPUT$(1,1)
3   PRINT In$;
4 WEND
5 CLOSE 1
```

Gibt den Inhalt von DESKTOP.INF aus.

EQV

Typ: Operator

Syntax: <num.Ausdruck> EQV <num.Ausdruck>

Erklärung: Verknüpft die beiden Ausdrücke bitweise äquivalent.

Beispiel:

```
0 PRINT BIN$((%1010 EQV %1100)+%10000)
1001
```

ERL

Typ: Funktion

Syntax: ERL

Erklärung: Gibt nach Verzweigen durch ON ERROR GOTO oder ON TRON GOSUB die Nummer der Zeile zurück, in der der Fehler auftrat, bzw. die zuletzt bearbeitet wurde.

ERR

Typ: Funktion

Syntax: ERR

Erklärung: Gibt nach Verzweigen durch ON ERROR GOTO die Fehler-Nummer an, nach Verzweigen durch ON TRON GOSUB die Nummer des gerade bearbeiteten Tokens.

ERR$

Typ: Funktion

Syntax: ERR$

Erklärung: Ergibt nach Verzweigen durch ON ERROR GOTO den Fehler-Meldungs-Text, nach Verzweigen durch ON TRON GOSUB den gerade bearbeiteten Befehl.

ERROR

Typ: Befehl

Syntax: ERROR(<num.Ausdruck>)

ERROR(<Fehlernummer>)

Erklärung: Erzeugt den durch den numerischen Ausdruck angegebenen Fehler.

Siehe auch Fehlerliste.

Beispiel:
```
ERROR 32
```

EXEC

Typ: Befehl

Syntax: EXEC <Stringausdruck>[,<Stringausdruck>[,<Stringausdruck>]]

EXEC <Dateiname>[,<Kommando>[,<Environment>]]

Erklärung: Startet ein beliebiges ausführbares Programm (".PRG", ".APP", ".TTP" ...). Optional kann auch eine Kommando-Zeile und ein Environment übergeben werden. Beim Kommando-String ist zu beachten, daß das erste Zeichen die Länge des restliche Kommando-Strings enthält und daß er nicht länger als 127 Zeichen wird.

Beispiel:
```
0 Cmd$="C:\TEST.BAS"
1 Cmd$= CHR$( LEN(Cmd$))+Cmd$
2 EXEC "C:\COMPILER.PRG",Cmd$
```

EXIT

Typ: Befehl

Syntax: EXIT {[<num.Ausdruck>]|TO <Marke>

EXIT {[Zahl der Strukturen>]|TO <Marke>

Erklärung: Mit EXIT kann eine Struktur - also eine Schleife, ein Unterprogramm oder ein SELECT-CASE - verlassen werden. Die Programmausführung wird direkt hinter dem Strukturende fortgesetzt. Auf diese Weise können also zusätzliche Abbruchbedingung in eine Schleife eingebaut oder eine Prozedur vorzeitig verlassen werden. Die Variante mit Anzahl zum Verlassen mehrerer Strukturen ist inkompatibel zum Compiler und sollte deshalb vermieden werden. Wird hinter TO noch ein Sprungziel angegeben, so wird nicht direkt am Strukturende weiter gemacht, sondern erst an der angegebenen Marke. Trotzdem darf auch hier nicht mehr als eine Struktur verlassen werden. Das Sprungziel muß also in der nächsthöheren Ebene liegen. Mit EXIT TO z.B. zwei Schleifen auf einmal zu verlassen ist verboten. Ebenso dürfen Prozeduren und mehrzeilige Funktionen nicht mit EXIT TO verlassen werden.

Beispiel:

```
0 REPEAT
1   INPUT "Geben Sie einen Wert ein
        (nur RETURN -› ENDE): ";W$
2   IF W$="" THEN EXIT
3   Summe!+=VAL(W$)
4 UNTIL 0
5 PRINT Summe!
```

EXP

Typ: Funktion

Syntax: EXP(<num.Ausdruck>)

Erklärung: Ergibt den Wert der Potenz von e (Eulersche Zahl) und dem numerischen Ausdruck. Die Umkehrfunktion ist LN.

Beispiel:

```
0 PRINT EXP(1.)
1 PRINT "Die Eulersche Zahl:"; EXP(1#)

 2.7182818
Die Eulersche Zahl: 2.7182818284590452
```

FACT

Typ: Funktion

Syntax: FACT(<num.Ausdruck>)

Erklärung: Berechnet die Fakultät des ganzzahlig-abgerundeten numerischen Ausdrucks. Der Definitionsbereich reicht von 0 - 1754; andere Werte erzeugen eine Fehlermeldung ("Overflow" bzw. "Illegal function call").

Beispiel:

```
0 PRINT FACT(1000)
1 PRINT FACT(10000)

 4.0238726E+2567
? Overflow in 1
```

FIELD

Typ: **Befehl**

Syntax: FIELD [#]<num.Ausdruck>,<num.Ausdruck> [AS <String-Variable>] [[,<num.Ausdruck> [AS <String-Variable>]]]

FIELD [#]<Dateinummer>,<Satzlänge> [AS <Puffervariable>] [[,<Satzlänge> [AS <Puffervariable>]]]

Erklärung: Definiert für die durch die Dateinummer gegebene Random-Access-Datei die Datenstruktur. Die Datei muß zuvor mit OPEN "R" geöffnet worden sein, die Datensatzlänge muß mindestens den summierten Satzlängen der FIELD-Anweisung entsprechen, darf also auch größer sein. Ist die Datenstruktur zu umfangreich, um in einer Zeile definiert zu werden, so kann sie in zwei Zeilen aufgeteilt werden. Man verwendet hierzu eine zweite FIELD-Anweisung, wobei die erste Anzahl ohne Puffervariable die Summe aller bereits aufgezählten repräsentiert

Nach einem Lese-Vorgang durch GET enthalten die Puffervariable den gelesenen Dateisatz; bei einem Schreib-Vorgang mit PUT werden die Inhalte der Puffervariablen als Datensatz gespeichert.

Es ist darauf zu achten, daß die Länge der Puffervariable zu keiner Zeit verändert wird. Hierzu können die Feldinhalte mit LSET oder RSET übergeben werden. Weiterhin sind als Puffervariable nur einfach Strings zugelassen - Feldelemente sind verboten.

Im Hinblick auf den Compiler gilt folgendes: Puffervariable für FIELD-Anweisungen dürfen niemals lokal verwendet werden (LOCAL), solange die FIELD-Anweisung noch aktiv ist. Sie sollten wirklich ausschließlich der Nutzung als Dateipuffer vorbehalten bleiben.

Beispiel:

```
0 OPEN "F",1,"C:\*.*",0
1 FIELD 1,21,1 AS Att$,2 AS Tim$,2 AS Dat$,4 AS Len$,14
                                                AS Name$
2 WHILE 1
3   GET 1,0: IF EOF(1) THEN EXIT
4   PRINT LEFT$(Name$, INSTR(Name$, CHR$(0))); TAB (15);
                                                  CVIL(Len$)
5 WEND
6 CLOSE 1
```

FILES

Typ: Befehl

Syntax: FILES [<Stringausdruck>]

FILES [<Dateiname>]

Erklärung: Gibt ein Inhaltsverzeichnis des in Dateiname genannten Pfades. Jokerzeichen innerhalb des Dateinames sind zugelassen.

FILESELECT

Typ: **Befehl**

Syntax: FILESELECT (<Stringvariable>,<Stringvariable>,<num. Variable>)

FILESELECT (<Pfad&Auswahl>,<Dateiname>,<OK>)

Erklärung: Ruft eine Dateiauswahl-Box auf. Es muß unbedingt ein gültiger Pfad angegeben werden, da sonst ein Systemabsturz folgt. Der Dateiname wird als Vorgabe angezeigt.

Vor Ausführung des Befehls sollte unbedingt die Maus mit MOUSEON eingeschaltet werden.

Pfad und Dateiname enthalten nach Ausführung die gewählte Datei. OK ist ungleich 0 (=wahr), wenn die Dateiauswahl-Box durch Doppelklick auf eine Datei oder mit OK verlassen wurde. Bei einem Fehler oder Abbruch ist OK gleich 0.

Beispiel:

```
GEMDOS(Drive,$19) ' Aktuelles Laufwerk ermitteln
Path$=" "*64:Adr=LPEEK(SEGPTR+28)+LPEEK(VARPTR(Path$))
GEMDOS(,$47,HIGH(Adr),LOW(Adr),0) 'Aktueller Pfadname
Path$=LEFT$(Path$,INSTR(Path$+CHR$(0),CHR$(0))-1)
Path$=CHR$(65+Drive)+":"+Path$+"\*.BAS"
MOUSEON:FILESELECT(Path$,Name$,Okay):MOUSEOFF
Dateiname$=LEFT$(Path$,LEN(Path$)-
INSTR(MIRROR$(Path$)+"\","\"))+Name$
IF OKay AND Dateiname$<>"" THEN OPEN "I",1,Dateiname
```

FILL

Typ: Befehl

Syntax: FILL <X>,<Y>,<Umriß-Farbe>

Erklärung: Füllt eine Fläche ab der durch X und Y gegebenen Koordinate. Als Flächenrand gilt jeder Punkt, der die Umriß-Farbe hat. Gibt man

Umriß-Farbe=-1 an, so gilt jeder Punkt als Flächenrand, der eine andere Farbe hat, als der an der X,Y-Koordinate.

Gefüllt wird mit der durch FILL COLOR definierten Farbe und dem durch FILL STYLE definierten Füllmuster.

Die Befehle PBOX, PROX, PCIRCLE, PELLIPSE und PPOLYGON füllen eine Fläche schneller als der entsprechende Zeichenbefehl mit anschließendem FILL.

Ist mittels CLIP ein Bildfenster definiert, wird außerhalb dieses Bereiches nicht gefüllt.

FILL COLOR

Typ: Befehl

Syntax: FILL COLOR=<num.Ausdruck>

Erklärung: Der numerische Wert gibt die bei Füll-Befehlen verwendtete Farbe an. Je nach Auflösung ist ein Wert zwischen 0 und 1, 0 und 3, bzw. 0 und 15 möglich.

Siehe auch PALETTE.

FILL PATTERN

Typ: Befehl

Syntax: FILL PATTERN = <num. Variable> [[,<num. Variable>]]

FILL PATTERN = <Integer-Wort-Feldvariable>(<Feldoffset>)

[[,<Integer-Wort-Feldvariable>(<Feldoffset>)]]

Erklärung: Durch FILL PATTERN = wird das benutzter-definierbare Füllmuster festgelegt. Die Bitkombinationen für das Füllmuster befinden sich in den ersten 16 Worten der Integer-Wort-Feldes ab dem angegebenen Feldoffset. Ist mehr als eine Farbebene vorhanden, so ist für jede Farbebene ein weiteres Feld anzugeben. Benutzt wird das so definierte Füllmuster mit FILL STYLE = 4,1.

Beispiel:

```
0 DATA %0000000000000000 ' = ----------------
1 DATA %0000001111000000 ' = ------****------
2 DATA %0000110000110000 ' = ----**----**----
3 DATA %0001000000001000 ' = ---*--------*---
4 DATA %0010000000000100 ' = --*----------*--
5 DATA %0010000000000100 ' = --*----------*--
6 DATA %0100110000110010 ' = -*--**----**--*-
7 DATA %0100000100000010 ' = -*------*------*-
```

```
 8 DATA %0100000100000010 ' = -*------*------*-
 9 DATA %0100000110000010 ' = -*------**------*-
10 DATA %0010000000000100 ' = --*----------*--
11 DATA %0010001111000100 ' = --*---****---*--
12 DATA %0001000000001000 ' = ---*--------*---
13 DATA %0000110000110000 ' = ----**-----**----
14 DATA %0000001111000000 ' = ------****------
15 DIM Patt%(22)
16 FOR I=0 to 15: READ Patt%(I+7): NEXT I
17 FILL PATTERN = Patt%(7)' Index gibt Start der Daten an
18 FILL COLOR = 1: FILL STYLE = 4,0: MODE = 1
19 PBOX 50,50,100,100
```

FILL STYLE

Typ: Befehl

Syntax: FILL STYLE=<num.Ausdruck>,<num.Ausdruck>

FILL STYLE=<Füllart>,<Füllstil>

Erklärung: Wählt den bei Füll-Befehlen verwendeten Füllstil. Mit Füllart kann gewählt werden:

0: Nicht füllen

1: Komplett füllen

2: Punktmuster

3: Strichmuster

4: Durch FILL PATTERN festgelegtes Muster

Füllstil muß immer angegeben werden, hat aber nur bei Füllart 2 und 3 eine Bedeutung. Siehe auch Füllstil-Tabelle.

FIX

Typ: Befehl

Syntax: FIX(<num.Ausdruck>)

Erklärung: Rundet den numerischen Wert auf eine ganze Zahl ab. Im Gegensatz zu INT wird auch im negativen Bereich "abgerundet".

Das Gegenstück zu FIX ist FRAC.

Beispiel:

```
0 PRINT FIX( PI )
1 PRINT FIX(-12.5), INT(-12.5)
```

```
 3
-12      -13
```

FN

Typ: Funktion

Syntax: FN <Variable>[(<Ausdruck>[[,<Ausdruck>]])]

FN <Funktions-Variable>[(<Parameter>[[,Parameter]])]

Erklärung: Ruft eine durch DEF FN definierte Funktion auf und weist der Funktions-Variable den Funktionswert zu.

Die in der Definition genannten Parameter müssen übergeben werden.

Beispiel:

```
0 PRINT FN Dreifach(3)
1 PRINT FN Primzahl$(12)
2 END
3 DEF FN Dreifach(A)=3*A
4 DEF FN Primzahl$(A)
5   LOCAL N,P$="Primzahl"
6   FOR N=2 TO SQR(A)
7     IF A MOD N=0 THEN P$="keine Primzahl"
8   NEXT N
9 RETURN P$

 9
keine Primzahl
```

FOR ... TO ...[STEP] ... NEXT

Typ: Befehl

Syntax: FOR <num. Variable> = <num.Ausdruck> TO <num.Ausdruck> [STEP <num.Ausdruck>: <Befehle> : NEXT [<num. Variable>][[,<num. Variable>]]

FOR <Schleifenvariable> = <Startwert> TO <Endwert> [STEP <Schrittweite>: <Schleifeninhalt> : NEXT [<Schleifenvariable>]

Erklärung: Die wohl bekannteste Programmschleife. Der Schleifenvariable wird zu Beginn der Startwert zugewiesen, welcher sodann bei jedem neuen Durchlauf um die Schrittweite erhöht oder erniedrigt wird. Fehlt die Schrittweite wird sie automatisch auf eins festgelegt.

Mögliche Schleifenvariable sind: Alle einfachen Variablentypen, jedoch keine Feldelemente. Zu beachten ist ferner: Wenn die Schleifenvariable abwärts gezählt werden soll, ist unbedingt eine negative Schrittweite

anzugeben (z.B. -1). Wenn die Schleifenbedingung gleich zu Anfang nicht erfüllt ist (z.B. Startwert ist größer als Endwert bei positiver Schrittweite), dann wird die Schleife überhaupt nicht durchlaufen. Die FOR-NEXT-Schleife ist also eine abwesende Schleife.

Die Schleife wird geschlossen durch die NEXT-Anweisung wobei die Schleifenvariable weggelassen werden kann. Die NEXT Anweisung bezieht sich dann immer auf die letzte FOR-Anweisung.

Um die Schleife vorzeitig zu verlassen verwenden sie EXIT.

Beispiel:

```
Ø DIM A(1Ø)
1 FOR I=Ø TO 1Ø
2   A(I)=RND(2Ø):PRINT A(I)
3 NEXT I
4 Maximum=Ø
5 FOR I=Ø TO 1Ø: Maximum=MAX(Maximum,A(I)): NEXT I
6 PRINT "Die größte Zahl ist: ";Maximum

Die größte Zahl ist: ....
```

FORM_ALERT

Typ: Befehl

Syntax: FORM_ALERT (<num.Ausdruck>,<Stringausdruck>[,<Variable>])

FORM_ALERT (<Default>,<Warnmeldung>[,<Rückgabe>])

Erklärung: Stellt eine Warn-Box dar. Default gibt an, welcher der in Warnmeldung genannten Knöpfe durch RETURN ausgelöst werden kann. Soll dies für keinen gelten, so ist Default gleich 0 zu übergeben.

Die Warnmeldung ist folgendermaßen aufgebaut:

[Icon][Zeile 1|Zeile 2|Zeile 3|Zeile 4|Zeile 5|][Knopf1|Knopf2|Knopf 3]

Icon ist eine Zahl zwischen 0 und 3:

0: kein Icon

1: Ausrufezeichen

2: Fragezeichen

3: Stop-Symbol

andere Werte können zu Abstürzen führen.

Die Zeilen dürfen jeweils höchstens 30 Zeichen lang sein.

Die Knopfgröße orientiert sich am längsten Knopftext. (maximal 20 Zeichen)

Nach Ausführung enthält Rückgabe die Nummer des gewählten Knopfes.

Beispiel:

```
0 Datei$="A:\TEST.TXT"
1 MOUSEON
2 FORM_ALERT (1,"[2][Die Datei|"+Datei$+"|wird
      unwiderruflich gelöscht.][OK|Abbruch]",Datei_Ex)
3 MOUSEOFF
4 IF Datei_Ex=1 THEN KILL A$
```

FRAC

Typ: Funktion

Syntax: FRAC(<num.Ausdruck>)

Erklärung: Ergibt den Nachkommaanteil des numerischen Ausdrucks. Das Vorzeichen bleibt erhalten.

Das Gegenstück zu FRAC ist FIX.

Beispiel:

```
PRINT FRAC( PI ),FRAC(5.36), FRAC(-5.36)
 .14159265358979324       .36     -.36
```

FRE

Typ: Befehl

Syntax: FRE <num.Ausdruck>

FRE <Speicheradresse>

Erklärung: Gibt einen durch MEMORY reservierten Speicherbereich wieder frei. Als Speicheradresse ist die Adresse anzugeben, die beim Aufruf von MEMORY zurückgegeben wurde.

Beispiel:

```
0 Puffer=MEMORY(1024)
1 PRINT "Puffer angelegt bei: ";HEX$(Puffer)
2 FRE Puffer
```

FRE

Typ: Funktion

Syntax: FRE({<num.Ausdruck>|<Stringausdruck>})

FRE({<Dummy>|<Laufwerk>})

Erklärung: Wird ein numerischer Dummy angegeben, so ergibt die Funktion, den für den Benutzer freien Speicherbereich, nachdem der Variablenspeicher aufgeräumt wurde (garbage collection).

Der Wert des Dummys ist dabei ohne Belang.

Wird ein Leerstring (="") übergeben, so ergibt die Funktion den für den Benutzer freien Speicherbereich, ohne daß zuvor eine garbage collection durchgeführt wird.

Sonst wird der Stringausdruck als Laufwerksbezeichner interpretiert, und der auf diesem Laufwerk freie Speicherbereich ermittelt.

Beispiel:
```
PRINT FRE(0), FRE(""), FRE("C:"), FRE("C:\AUTO\")
```

FSEL_INPUT

Typ: Befehl

Syntax: siehe FILESELECT

Erklärung: siehe FILESELECT

GEMDOS

Typ: Befehl

Syntax: GEMDOS[(][<Variable>],<num.Ausdruck>[[,[L]<num.Ausdruck>]][)]GEMDOS[(][<Rückgabe-Variable>],

<Funktionsnummer>[[,[L]<Parameter>]][)]

Erklärung: Ruft eine Funktion des GEMDOS (ein Teil des Betriebssystems) auf.

Wenn vor den Parametern ein "L " gestellt ist, so wird der Parameter als LONG übergeben, ansonsten immer als WORD.

Beispiel:
```
0 PRINT FN Get_Path$
1 END
2 DEF FN Get_Path$' Standard-Pfad ermitteln
3  LOCAL Path$,Drive,Adr
4   GEMDOS (Drive,$19)' Standard-Laufwerk
5   Path$= CHR$(0)*128
6   Adr= LPEEK( VARPTR(Path$))+ LPEEK( SEGPTR +28)
7   GEMDOS (,$47,L Adr,Drive+1)
8   Path$= CHR$(Drive+65)+":"+ LEFT$(Path$, INSTR(Path$,
                                          CHR$(0))-1)+"\"
9  RETURN Path$
10 END_FN
```

GET

Typ: Befehl

Syntax: GET <num.Ausdruck>[,{<num.Ausdruck>|<Str.Var}],<num.Ausdruck>

1: GET <Dateinummer>,<Satznummer>

2: GET <Dateinummer>,<Speicheradresse>,<Anzahl>

3: GET <Dateinummer>,<String-Variable>,<Anzahl>

Erklärung: Nach Syntax 1 wird ein Datensatz aus der durch die Dateinummer gegebenen Datei gelesen und den in FIELD genannten Puffervariablen zugewiesen. Die Datei muß zuvor mittels OPEN "R" geöffnet worden sein. Ist der Datensatz nicht vorhanden enthalten die Puffervariablen Leerzeichen, EOF gibt den Wert -1 (=wahr) zurück.

Nach Syntax 2 und 3 wird aus der durch die Dateinummer gegebenen Datei die gegebene Anzahl Zeichen gelesen und ab der Speicheradresse bzw. in der String-Variable abgelegt. Die Datei muß zuvor mit OPEN "U" geöffnet worden sein. Gelesen wird ab der aktuellen Dateiposition, die mit SEEK gesetzt werden kann.

Beispiel:

```
0 OPEN "U",1,"BILDSCH.PIC" ' BILDSCH.PIC muß im aktuellen
  Verzeichnis vorhanden sein
1 XBIOS Adresse,3 ' holt Bildschirm-Adresse
2 GET 1,Adresse,32000 ' nicht bei Großbildschirmen
3 CLOSE 1
```

GOSUB

Typ: Befehl

Syntax: GOSUB <Marke>

Erklärung: Verzweigt in das durch <Marke> definierte Unterprogramm.

Vom Unterprogramms kann mittels RETURN wieder in das Hauptprogramm zurückgesprungen werden. Dann wird als nächstes der der GOSUB-Anweisung folgende Befehl ausgeführt.

Die im Unterprogramm verwendeten Variablen sind global (siehe LOCAL). Unter OMIKRON.Basic geschriebene Programm können im Prinzip vollständig auf GOSUB verzichten, da ein Unterprogrammaufruf immer auch mittels einer Prozedur verwirklicht werden kann. Ein Unterprogramm per GOSUB aufzurufen ist eigentlich nicht mehr zeitgemäß. Aus Gründen der Kompatibilität wurde der Befehl im Sprachumfang belassen.

GOTO

Typ: Befehl

Syntax: GOTO <Marke>

Erklärung: Verzweigt an die durch <Marke> definierte Stelle im Programm. ACHTUNG: Verzweigen Sie niemals in eine Struktur (Schleife oder SELECT-CASE) oder in eine Prozedur. Sprünge per GOTO sind nur in der gleichen Hierachie-Ebene erlaubt. Um eine Struktur vorzeitig zu verlassen verwenden Sie EXIT.

Generell führt die Verwendung von GOTO sehr schnell zu sehr unübersichtlichen Programmen. Man sollte daher zu viele GOTOs nach Möglichkeit vermeiden. Die strukturierte Programmierung, wie sie von OMIKRON.Basic unterstützt wird, kommt in jedem Falle auch ganz ohne GOTO aus. Es ist jedoch nicht auszuschließen, daß in einem oder anderen Fall (z.B. bei nicht behebbaren Fehlern) ein GOTO einfach praktischer ist.

Siehe auch RENUM.

Beispiel:

```
Zeilennummer:    GOTO 36
Berechnung:      GOTO 5*(X-3)
Label:           GOTO Abfrage
String:          A$="52": GOTO A$
                 B$="Loop": GOTO B$
```

HCOPY

Typ: Befehl

Syntax: HCOPY

Erklärung: Ruft die Systemroutine "Hard-Copy" auf, die auch über Alternate Help erreichbar ist. Die Einstellung auf die verschiedenen Druckertypen erfolgt über das Kontrollfeld-Accessory, das dem Atari beiliegt.

Wie in der Systemroutine kann der Ausdruck durch Alternate Help abgebrochen werden.

Beispiel:

```
HCOPY
```

HCOPY TEXT

Typ: Befehl

Syntax: HCOPY TEXT

Erklärung: Gibt eine Textkopie des Bilddschirms auf dem Drucker aus.

WICHTIG: Diese Funktion wird vom COMPILER ab Version 3.5 nicht mehr unterstützt (Kompatibilitätsprobleme).

Beispiel:
```
HCOPY TEXT
```

HEIGHT

Erklärung: siehe TEXT HEIGHT

HELP

Erklärung: siehe ON HELP GOSUB

HEX$

Typ: Funktion

Syntax: HEX$(<num.Ausdruck>)

Erklärung: Wandelt den numerischen Ausdruck in eine Zeichenkette um, die den gerundeten Wert des Ausdrucks als Hexadezimalzahl darstellt.

Beispiel:
```
PRINT HEX$(255), HEX$(-255), HEX$(-1)

FF          -FF         -1
```

HIGH

Typ: Funktion

Syntax: HIGH(<num.Ausdruck>)

Erklärung: Ermittelt die oberen 16 Bit des in Integer-Langwort gewandelten numerischen Ausdrucks. Gegenstück zu dieser Funktion ist LOW.

Beispiel:
```
PRINT HEX$(HIGH($12346789)),HIGH(-1)

1234        -1
```

H_CHAR

Typ: Funktion

Syntax: H_CHAR

Erklärung: Ergibt die Höhe des Bildschirms in Zeichen. Der Wert entspricht den Informationen, die vom VDI bei Inquire Character Cells gelifert werden.

Beispiel:

```
(für Auflösung ST HOCH)
0 PRINT W_CHAR;" Zeichen pro Zeile"
1 PRINT H_CHAR;" Zeichen pro Spalte"
2 PRINT W_PIXEL;" Punkte ist der Bildschirm breit"
3 PRINT H_PIXEL;" Punkte ist der Bildschirm hoch"


80 Zeichen pro Zeile
25 Zeichen pro Spalte
640 Punkte ist der Bildschirm breit
400 Punkte ist der Bildschirm hoch
```

H_PIXEL

Typ: Funktion

Syntax: H_PIXEL

Erklärung: Ergibt die Höhe des Bildschirms in Bildpunkten (Pixels). Der Wert entspricht den Informationen, die vom VDI bei V_Openvwk geliefert werden. (Siehe auch H_CHAR).

IF ... THEN ... ELSE ... ENDIF

Typ: Befehl

Syntax: IF <num.Ausdruck> THEN <Befehle> [ELSE <Befehle>] [ENDIF]

IF <log. Ausdruck> THEN <abhängige Befehle (wahr) [ELSE <abhängige Befehle (falsch)>] [ENDIF]

Erklärung: Die IF-THEN-Anweisung erlaubt die bedingte Ausführung von Befehlen. Die von der THEN-Anweisung abhängigen Befehle werden nur ausgeführt, wenn der logische Ausdruck wahr (ungleich 0) ist. Die von der ELSE-Anweisung abhängigen Befehle entsprechend bei falschem logischen Ausdruck (gleich 0).

Wenn alle abhängigen Befehle in eine Zeile passen, kann das ENDIF entfallen. Ansonsten beschließt das ENDIF den Block der abhängigen

Anweisungen. Stehen sehr viele Alternativen zur Auswahl, so ist in vielen Fällen die Verwendung von SELECT CASE vorteilhafter.

Wenn direkt vor oder nach dem THEN ein Befehlstrenner kommt (Befehlstrenner sind z.B. neue Zeile, ELSE, ENDIF, UNTIL, WEND, '), wird die IF-Konstruktion mehrzeilig.

Eine IF-Konstruktion kann an jeder Stelle mit EXIT verlassen werden.

Beispiele: a) einfachster Fall einzeilig:

```
IF ‹Bedingung› THEN ...
```

b) einzeilig mit ELSE:

```
IF ‹Bedingung› THEN ... ELSE ...
```

c)einfachster Fall mehrzeilig:

```
IF ‹Bedingung› THEN
  ...
ENDIF
```

d) mehrzeilig mit ELSE:

```
IF ‹Bedingung› THEN
  ...
ELSE
  ...
ENDIF
```

so geht's auch:

```
IF ‹Bedingung›
THEN ...
ELSE ...
ENDIF
```

e) natürlich auch schachtelbar:

```
IF ‹Bedingung1› THEN
  IF ‹Bedingung2› THEN
    ...
  ENDIF
ELSE IF ‹Bedingung› THEN ...
  ...
 ENDIF
```

IMP

Typ: Operator

Syntax: <num.Ausdruck> IMP <num. Ausruck>

Erklärung: Die beiden Aussagen werden logisch implement verknüpft. Die Wahrheitstabelle hierzu führt nur zu einer unwahren Aussage, wenn der erste Ausdruck wahr, er zweite aber unwahr ist.

Beispiel:

```
Ø PRINT BIN$((%1Ø1Ø IMP %11ØØ)+%1ØØØØ)
11Ø1
```

INKEY$

Typ: Funktion

Syntax: INKEY$

Erklärung: Holt eine Eingabe vom Tastatur-Buffer ab. Ist keine Eingabe vorhanden, gibt die Funktion einen Leerstring. Sonst ist der String 4 Zeichen lang.

Der ASCII-Wert des ersten Zeichens ergibt den Zustand der Shift-Tasten:

Bit 0: rechtes Shift

Bit 1: linkes Shift

Bit 2: Control

Bit 3: Alternate

Bit 4: Caps Lock

Der ASCII-Wert des zweiten Zeichens entspricht dem Scan-Code der gedrückten Taste (siehe auch Scan-Code-Tabelle).

Das dritte Zeichen hat keine Bedeutung; ist immer CHR$(0).

Das vierte Zeichen entspricht dem ASCII-Wert der Eingabe, oder ist CHR$(0), wenn die Eingabe keine ASCII-Wert hat (z.B. Home)

Beispiel:

```
0 REPEAT
1   REPEAT
2     A$= INKEY$
3   UNTIL LEN(A$)
4   FOR N=1 TO 4
5     PRINT ASC( MID$(A$,N,1)),
6   NEXT
7     IF RIGHT$(A$,1)>=" " THEN PRINT RIGHT$(A$,1) ELSE
                                                    PRINT
8 UNTIL ASC( MID$(A$,2,1))=97 'Ende mit Undo
```

```
 0      30      0      97     a
 0      48      0      98     b
 2      46      0      67     C
 2      32      0      68     D
 0       1      0      27
 0      28      0      13
 0      97      0       0
```

Beispiel:

```
 0 REPEAT
 1   REPEAT
 2     A$=INKEY$
 3   UNTIL A$<>"" 'Bis Taste gedrückt
 4   A=CVIL(A$)
 5   Shift=A SHR 24: Scan=A SHR 16 AND $FF
 6   Ascii=A AND $FF
 7   IF Ascii<>0 THEN PRINT "Zeichen ";CHR$(Ascii)
 8   IF Scan=$48 THEN PRINT "Taste Pfeil hoch"
 9   PRINT "Taste mit Tastennummer ";Scan
10 UNTIL Ascii=32 ' Bis <Space> gedrückt wird
```

INLINE

Typ: Befehl

Syntax: INLINE <String>

Erklärung: führt Maschinensprachebefehle aus.

Beispiel: `INLINE "A009"`

ruft LINE-A Nr. 9 auf, die Show-Mouse-Funktion. Der String enthält den INLINE-Cde in hexadezimaler Form

INPUT

Typ: Befehl

Syntax: INPUT [<Stringausdruck>;[<Stringausdruck>;...]<Variable>[,<num. Variable>...]

Erklärung: Der INPUT Befehl liest eine oder mehrere Variablen von der Tastatur ein. Wenn angegeben, wird eine Eingabeaufforderung (Prompt) ausgegeben, ansonsten einfach ein Fragezeichen. Die Eingabe wird mit [RETURN] abgeschlossen. Wenn mehrere Werte in einer Zeile eingegeben werden sollen, müssen diese durch Kommata abgetrennt werden. Wenn num. Variable eingelesen werden sollen, wird alles bis zum ersten ungültigen Zahlzeichen als Wert übernommen, der Rest wird ignoriert. Führende Leerzeichen werden überlesen. Der Wert, den die Variable vor Ausführung des INPUT-Befehls hatte, spielt keine Rolle: Es wird im jedem Fall ein neuer Wert zugewiesen. Wesentlich erweiterte Möglichkeiten bietet der INPUT USING Befehl.

Beispiel:

```
0 INPUT "Geben Sie 3 Werte ein: ";A,B,C
1 PRINT A,B,C
2 INPUT @(10,0);"Geben Sie Ihren Namen ein: ";N$
3 PRINT N$
```

INPUT

Typ: Befehl

Syntax: INPUT #<num.Ausdruck>,<Variable>[[,<Variable>]]

INPUT #<Dateinummer>,<Variable>[[,<Variable>]]

Erklärung: Es werden aus einer sequentiellen Datei eine oder mehrere Variablen eingelesen. Wie bei INPUT können mehrere Variablen in einer Zeile, durch Komma getrennt, erscheinen.

WICHTIG: Mit INPUT # müssen Sie immer eine ganze Zeile der Datei mit einer INPUT Anweisung lesen, d.h. die INPUT Anweisung muß genauso viel Variablen enthalten, wie Werte in einer Zeile stehen. Wenn Sie nicht wissen, wieviel Werte in einer Zeile stehen, so lesen Sie besser mit LINE INPUT die ganze Zeile ein und weisen die Werte erst später an die eizelnen Variablen zu.

Beispiel:

```
0 OPEN "O",1,"C:\TEST.DAT"
1   WRITE #1,1,2,3
2   WRITE #1,4,5,6
3 CLOSE 1
4 '
5 OPEN "I",1,"C:\TEST.DAT"
6   WHILE NOT EOF(1)
7     INPUT #1,A,B,C
8     PRINT A,B,C
9   WEND
10 CLOSE 1
```

INPUT USING

Typ: Befehl

Syntax: INPUT [<Stringausdruck>;]<String-Variable> USING [<Stringausdruck>],[<num.Variable>],[<num.Ausdruck>],

[<num.Ausdruck>][,<num.Variable>]

INPUT [<Prompt>;]<Eingabe-Variable> USING

[<Steuerstring>],[<Rückgabe-Variable>],[<Länge>],
[<Füllzeichen>][,<Positions-Variable>]

Erklärung: INPUT USING ermöglicht eine formatierte Maskeneingabe mit diversen Einstellungsmöglichkeiten. Abhängig vom Steuerstring werden bei der Eingabe nur bestimmte Zeichen zugelassen:

- "0" Ziffern
- "à" Buchstaben (einschl. länderspezifische Zeichen)
- "a" Buchstaben (ohne länderspezifische Zeichen)
- "%" Sonderzeichen (ausschl. länderspezifische Zeichen)
- "^" Ctrl-zeichen (Eingabe mit [Control]-[A] [Control]-[Buchstabe])
- "+<Zeichen>" einzelnes Zeichen zulassen
- "-<Zeichen>" einzelnes Zeichen verbieten

Ob Sie dabei die Steuerzeichen selbst groß oder klein schreiben, ist egal.
Um also Namen eingeben zu können, könnte ihr Steuerstring so aussehen: "A +- +." alle Buchstaben einschl. Sonderzeichen, der Bindestrich und der Punkt sind zugelassen.
Telefonnummern würde man mit "0 +/" eingeben können.
Weiterhin besteht die Möglichkeit bestimmte Zeichen gleich bei der Eingabe umwandeln zu lassen:

- "u" alles nach Großbuchstaben wandeln
- "l" alles nach Kleinbuchstaben wandeln
- "c<Zeichen1><Zeichen2>" wird Zeichen1 eingegeben, so wird es automatisch durch Zeichen2 ersetzt. Hinweis: Zeichen1 muß natürlich auch in der Auswahl der zugelassenen Zeichen sein.

Um also bei einer Zahleneingabe immer sofort statt Dezimalpunkt mit Komma zu arbeiten würde ein Steuerstring "0 +.+, c.," benötigt.
Grundsätzlich wird die Eingabe mit RETURN beendet. Sie können jedoch mit dem Steuerstring weitere Tasten oder Ereignisse bestimmen, die die Eingabe beenden:

- "x<Zeichen>" Sobald die Taste mit demselben ASCII-Code wie <Zeichen> gedrückt wird, wird die Eingabe abgebrochen.
- "s<Zeichen>" Sobald die Taste mit demselben Scancode wie der ASCII-Code von <Zeichen> gedrückt wird, wird die Eingabe abgebrochen. Dies ermöglicht z.B. die Unterscheidung zwischen Ziffernblock und Haupttastatur und erlaubt auch den Abbruch durch Tasten, die als ASCII-Code eine Null zurückgeben, wie z.B. [Cursor hoch].
- "<" linke Randüberschreitung: Die Eingabe wird abgebrochen, wenn der Cursor über den linken Rand hinaus bewegt wird.
- ">" rechte Randüberschreitung: Die Eingabe wird abgebrochen, wenn der Cursor über den rechten Rand hinaus bewegt wird.
- "m" die INPUT USING darf durch sogenannte Multitasking-Befehle wie ON TIMER GOSUB oder ON MOUSEBUT GOSUB unterbrochen

werden. Normalerweise ist INPUT USING ein Befehl und kann nicht durch Multitasking unterbrochen werden. Wenn der Steuerstring ein "m" enthält wird dies trotzdem ermöglicht.

Hinweis: In Compilaten ist eine Unterbrechung des INPUT USING Befehls grundsätzlich immer möglich. Wenn die Unterbrechung allerdings mehr erledigen soll als nur eine Kleinigkeit sollte vorübergehend die INPUT USING Funktion verlassen werden. Dies wird mit POKE RESERVED(4),1 ausgelöst.

Der Prompt wird wie bei allen Eingabeanweisungen vor der Eingabe ausgegeben und kann natürlich auch ein Positionangabe mit "@" enthalten.

Die Eingabe-Variable muß immer von Stringtyp sein, da sie während der Eingabe als Puffer dient. Numerische Eingaben werden nach der Eingabe mittels VAL umgewandelt.

Die Rückgabe-Variable gibt Aufschluß über die Abbruchursache:

0 die Eingabe wurde mit RETURN verlassen

-1 linke Randüberschreitung

-2 rechte Randüberschreitung

-3 Unterbrechung durch COMPILER: RESERVED(4)
INTERPRETER: MULTITASKING

Positive Werte stehen für eine andere Abbruchtaste. Sie wird durch einen vier Byte langen Wert gekennzeichnet, der - ähnlich wie bei INKEY$ die Shift(b)its, den (S)cancode und den (A)SCII-Code enthält: $BB SS 00 AA (hexadezimale Darstellung)

Durch die Angabe einer Länge kann die Eingabe auf eine bestimmte Zeichenzahl begrenzt werden. Die Länge sollte immer mit angegeben werden, da eine zu lange Eingabezeile (mehr Spalten als darstellbar) nicht vernüpftig funktioniert.

Der ASCII-Code des Füllzeichens bestimmt das Zeichen, mit dem die Eingabezeile hinterlegt wird, um das Eingabefeld anzudeuten. Wird nichts angegeben, so ist das "_"-Zeichen voreingestellt. Denkbar wäre z.B. 42 ("*") für Zahleneingaben bei Schecks oder 32 (Leerzeichen), wenn gar nichts hinterlegt werden soll.

Die Positions-Variable schließlich bestimmt die Position, die der Cursor zu Beginn der Eingabe einnimmt. Sie liefert auch die zuletzt eingenomme Cursor-Position zurück. Man kann also z.B. den Cursor gezielt an die Fehlerstelle bewegen, wenn im Fehlerfall die Eingabe wiederholt werden muß. Es wäre auch denkbar, den Cursor immer wieder an den Anfang der Eingabe zu stellen, um so schneller Korrekturen vornehmen zu können.

WICHTIG: Die Eingabe-Variable wird im Unterschied zu INPUT nicht in jedem Fall zuerst gelöscht, sondern erscheint als Vorgabe in der Eingabezeile. Damit können Sie dem Anwender bereits sinnvolle Eingaben vorschlagen z.B. das aktuelle Datum. Es genügt dann ein einfaches RETURN, um die Eingabe zu übernehmen. Wenn mehrfach dieselben oder ähnliche Eingaben verlangt werden, kann der alte Wert einfach stehen bleiben oder geringfügig modifiziert übernommen werden.

Hinweis: Wenn wärend der INPUT USING Anweisung Control-C gedrückt wird, wird erst nach verlassen der Eingabe das Programm abgebrochen. Nach Beendigung der Eingabe mit INPUT USING wird kein Zeilenvorschub ausgegeben. Soll also die nächste Ausgabe in einer neuen Zeile erfolgen, so muß zuvor ein Zeilenvorschub ausgegeben werden.

Beispiel:

```
Ø Eingabe
 1 END
 2 DEF PROC Eingabe
 3  CLS
 4  PRINT @(5,5);"Name    :";
 5  PRINT @(6,5);"Straße  :";
 6  PRINT @(8,5);"PLZ/Ort :";
 7  PRINT @(1Ø,5);"Telefon :";
 8  PRINT @(13,5);"Angaben in Ordnung (J/N) ?";
 9  Exit$="s"+ CHR$($48)+"s"+ CHR$($5Ø)
1Ø  Max_Feld=5
11  REPEAT
12    SELECT Feld
13      CASE Ø
14        INPUT @(5,15);Name$ USING "a+-+
            "+Exit$,Taste,5Ø
15      CASE 1
16        INPUT @(6,15);Strasse$ USING "aØ+
            +-"+Exit$,Taste,3Ø
17      CASE 2
18        INPUT @(8,15);Plz$ USING "Øx "+Exit$,Taste,4
19      CASE 3
2Ø        INPUT @(8,2Ø);Ort$ USING "aØ+
      +-+/"+Exit$,Taste,5Ø
21      CASE 4
22        INPUT @(1Ø,15);Tel$ USING "Øc-/+/+
```

```
                "+Exit$,Taste,15
23      CASE 5
24        Jn$=""
25        INPUT @(13,32);Jn$ USING "u+J+N"+Exit$,Taste,1
26    END_SELECT
27    Scan= HIGH(Taste) AND $FF
28    IF Scan=$48 THEN Feld= MAX(Feld-1,0)'  Cur Up
29    IF Scan=$50 OR Taste=0 THEN Feld=
        MIN(Feld+1,Max_Feld)'  Cur Down
30  UNTIL Jn$="J"
31 RETURN
```

INPUT$

Typ: Funktion

Syntax: INPUT$(<num.Ausdruck>[,<num.Ausdruck>])

INPUT$(<Anzahl Zeichen>[,<Dateinummer>])

Erklärung: Liest eine bestimmte Anzahl Zeichen von der Console (der Tastatur) bzw. von der genannten Datei.

Beispiel:

```
0 OPEN "I",1,"C:\DESKTOP.INF"
1   WHILE NOT EOF(1)
2     PRINT INPUT$(1,1);
3   WEND
4 CLOSE 1
5 '
6 PRINT "Taste drücken ..."
7 Dummy$= INPUT$(1)
8 END
```

INSTR

Typ: Funktion

Syntax: INSTR([<num.Ausdruck>,]<Stringausdruck>,<Stringausdruck>)

INSTR([<Suchbeginn>,]<Stringausdruck>,<Zeichenfolge>)

Erklärung: Sucht die angegebene Zeichenfolge im Stringausdruck und gibt dessen Position zurück. Ist die Zeichenfolge nicht im Stringausdruck enthalten, so ist der Funktionswert 0.

Der ganze Stringausdruck wird vom ersten Zeichen ab durchsucht, es kann aber auch mit Suchbeginn eine Startposition übergeben werden.

Das erste Zeichen im Stringausdruck hat die Position 1.

Beispiel:

```
PRINT INSTR("OMIKRON.Software",".")
0 Pfad$="C:\PROJEKTE\BASIC\SOURCES\"
1 P=INSTR(Pfad$,"\")
2 WHILE P
3   PRINT "Backslash gefunden an Position:"; P
4   P=INSTR(P,Pfad$,"\")
5 WEND

 8
Backslash gefunden an Position: 3
Backslash gefunden an Position: 12
Backslash gefunden an Position: 18
Backslash gefunden an Position: 26
```

INT

Typ: Funktion

Syntax: INT(<num.Ausdruck>)

Erklärung: Liefert die nächst kleinere ganze Zahl. Dadurch werden positive Werte ab- und negative Werte "auf"gerundet.

Die Funktion FIX rundet immer ab.

Beispiel:

```
PRINT INT( PI ), INT(12.5), INT(-12.5)

 3         12       -13
```

INV

Erklärung: siehe MAT

IPL

Typ: Befehl

Syntax: IPL <num.Ausdruck>

Erklärung: Weist dem System den numerischen Ausdruck ls Interrupt-Priority-Level zu. Alle Interrupts deren Level gleich oder kleiner ist, als der gegebene werden dann ignoriert. Ein Programmabbruch durch Control C kann verhindert werden, indem das höchstwertige Bit (=31) gesetzt wird.

Standardeinstellung ist IPL 3.

Beispiel:
```
Ø IPL 3+(1 SHL 31) 'verhindert Programmabbruch mit CTRL-C
```

ACHTUNG: Mit diesem Befehl können sämtliche Interrupts des Rechners gesperrt werden. In der Regel können Sie durch Angabe von IPL 3 den Interrupt-Zustand wieder normalisieren. Sie können mit diesem Befehl aber auch z.B. den Tastaturinterrupt sperren (IPL 6) in diesem Fall sind kein Eingaben von der Tastatur mehr möglich. Sie können dann den Zustand nicht mehr normalisieren und müssen den Rechner neu starten. Verwenden Sie daher diesen Befehl nur äußerst vorsichtig!!

JOYSTICK

Typ: Funktion

Syntax: A= JOYSTICK(<Joysticknummer>)

Erklärung: Bevor Sie die JOYSTICK-Funktion benutzen können, müssen Sie erst dem Tastaturprozessor mitteilen, daß jetzt ein Joystick angeschlossen ist und die eintreffenden Daten am Joystick/Mausport nicht mehr von der Maus kommen. Dies geschieht mit BIOS ,3,4,20. Mit BIOS ,3,4,8 wird die Maus wieder eingeschaltet.

Beispiel:
```
Ø BIOS ,3,4,20
1 REPEAT
2   J=JOYSTICK(Ø)
3   IF BIT(7,J) THEN PRINT "Feuer!!! ";
4   IF BIT(3,J) THEN PRINT "Rechts ";
5   IF BIT(2,J) THEN PRINT "Links ";
6   IF BIT(1,J) THEN PRINT "Hoch ";
7   IF BIT(Ø,J) THEN PRINT "Runter ";
8 UNTIL LEN( INKEY$)
9 BIOS ,3,4,8
```

KEY

Typ: Befehl

Syntax: KEY <num.Ausdruck>=<Stringausdruck>

Erklärung: Weist den Funktionstasten F1 bis F10 (Nummer 1 bis 10), bzw. Shift-F1 bis Shift-F10 (Nummer 11 bis 20) einen maximal 32 Zeichen langen Stringausdruck zu.

Auch Steuerzeichen wie Cursor hoch o.ä. lassen sich über die entsprechenden Escape-Sequenzen (siehe VT-52 Tabelle) eingeben.

Escape kann durch vorhergehendes [Control]-[A] in den Stringausdruck aufgenommen werden:

KEY1="[Control]-[A] [Escape] [C]".

Return erhält man durch CHR$(13):

KEY2="stop"+CHR$(13)

Die aktuellen Funktionstastendefinitionen können über SAVE SETTINGS im Menü MODE abgespeichert werden. Eine nichtdefinierte Funktionstaste liefert den ASCII-Code 0 zurück.

Siehe auch KEY LIST.

KEY LIST

Typ: Befehl

Syntax: KEY LIST

Erklärung: Gibt die Liste der Funktionstasten F1 bis F10 (Nummer 1 bis 10) und Shift-F1 bis Shift-F10 (Nummer 11 bis 20) zusammen mit den über KEY definierten Texten aus.

Beispiel:

```
KEY LIST

KEY  1="OMIKRON.Software"
KEY  2=""
.....
```

KILL

Typ: Befehl

Syntax: KILL <Stringausdruck>

KILL <Dateiname>

Erklärung: Die genannte Datei wird gelöscht.

Beispiel:

```
KILL "TEST.BAS"
```

LDUMP

Typ: Befehl

Syntax: LDUMP [{Buchstabe|Buchstabe-Buchstabe}[[,{Buchstabe|Buchstabe-Buchstabe}]]]

Erklärung: Alle Variablen werden mit Inhalt auf dem Drucker ausgegeben. Bei Feldern wird jedoch nur der Dimensionsbereich angegeben.

Optional können einzelne Buchstaben oder Buchstabenbereiche (nach Alphabet) genannt werden. Dann werden nur solche Variablen aufgelistet, die mit einem der genannten Buchstaben beginnen.

LEFT$

Typ: Funktion

Syntax: LEFT$(<Stringausdruck>,<num.Ausdruck>)

Erklärung: Ergibt einen Teilstring des Stringausdrucks beginnend mit dem ersten Zeichen mit der durch den numerischen Ausdruck gegebenen Länge. Ist der numerische Ausdruck größer als die Länge des Stringausdrucks, so wird der gesamte String zurückgegeben.

Siehe auch RIGHT$, MID$.

Beispiel:

```
Ø A$="OMIKRON.Software"
1 PRINT LEFT$(A$,7)

OMIKRON
```

LEN

Typ: Funktion

Syntax: LEN(<Stringausdruck>)

Erklärung: Ergibt die Länge des Stringausdrucks (maximal 32766).

LET

Typ: Befehl

Syntax: LET <Variable>=<Ausdruck>

Erklärung: Weist der Variablen den Ausdruck zu. Auf den Befehl kann aber verzichtet werden (er ist nur aus Kompatibilitätsgründen implementiert).

Die Zuweisung lautet dann:

<Variable>=<Ausdruck>

Beispiel:

```
Ø LET A=5+3
1 LET A*=2
2 PRINT A

16
```

LIBRARY

Typ: Befehl

Syntax: LIBRARY <Library-Name>, <Pfad & Dateiname>

Erklärung: Der LIBRARY-Befehl lädt eine OMIKRON.BASIC- Befehlserweiterung (im folgenden "Library" genannt) in Ihr Programm, falls es die betreffende Library noch nicht enthält. Der erste Parameter ist der Library-Name, der zweite zeigt an, wo die Library auf dem Massenspeicher zu finden ist.

Beispiel:

```
LIBRARY Easygem,"C:\EASYGEM.LIB"
```

Wenn die Library noch nicht in Ihrem Programm enthalten ist, wird Sie mit diesem Kommando nachgeladen. Anschließend steht die Library in Zeile Nummer 65534 Ihres Programms. Steht die Library einmal dort, so braucht Sie auch bei erneuten Programmstarts nicht mehr nachgeladen zu werden. Die gesamte Library ist praktisch in eine Zeile gepackt im Programm enthalten. Sie benötigt natürlich wesentlich mehr Speicherplatz als eine gewöhnliche andere Programmzeile.

EasyGEM würde z.B. so aussehen:

```
65534 LIBRARY CODE Easygem
```

Diese Zeile belegt mehr als 64 KB BASIC-Speicher. Wenn sie diese Zeile löschen, haben Sie EasyGEM aus dem Speicher entfernt. Das Laden einer Library geht viel schneller, als es z.B. mit dem MERGE-Befehl möglich wäre. Das Laden von EasyGEM dauert z.B. nur ungefähr 10 Sekunden.

LIBRARY CODE

Typ: Dieses Wort ist reserviert. Bitte nicht verwenden.

LINE COLOR

Typ: Befehl

Syntax: LINE COLOR=<num.Ausdruck>

Erklärung: Der numerische Wert gibt die bei Zeichen-Befehlen verwendtete Farbe an. Siehe auch Palette.

LINE INPUT

Typ: Befehl

Syntax: LINE INPUT [#<num.Ausdruck>][<String-Ausdruck>;]<Variable>[[,<Variable>]]

Erklärung: LINE INPUT entspricht fast in jeder Hinsicht dem normalen INPUT. Es kann sogar wie INPUT mit USING zusammen zur Masken-Eingabe verwendet werden (siehe INPUT USING). Lediglich Trennzeichen wie z.B. Kommas werden von LINE INPUT ignoriert; es wird immer bis zum Zeilenende eingelesen. Wenn also mehr als eine Variable eingelesen werden soll, so sind diese in unterschiedlichen Zeilen einzugeben.

Beispiel:

```
0 OPEN "O",1,"C:\TEST.DAT"
1  WRITE #1,"123","456","487"
2  WRITE #1,"123","456","487"
3 CLOSE 1
4 OPEN "I",1,"C:\TEST.DAT"
5   INPUT #1,A$,B$,C$
6   LINE INPUT #1,Line$
7 CLOSE 1
8 PRINT A$,B$,C$
9 PRINT Line$

123       456       487
"123","456","487"
```

LINE PATTERN

Typ: Befehl

Syntax: LINE PATTERN = <num.Ausdruck>

LINE PATTERN = <Linienmuster>

Erklärung: Das Linienmuster ergibt, als 16 Bit Integer-Wert aufgefaßt, das Linienmuster. Jedes gesetzte Bit wird gezeichnet, jedes gelöschte bleibt frei.

Beispiel: LINE PATTERN = $FFFF ' durchgezogen

LINE PATTERN = $FF00 ' gestrichelt 50 %

LINE PATTERN = $F000 ' gestrichelt 25 %

LINE STYLE

Typ: Befehl

Syntax: LINE STYLE=<num.Ausdruck>

Erklärung: Der numerische Wert gibt die bei Zeichen-Befehlen verwendtete Linienart an. Folgende Linienarten sind möglich:

1: durchgezogene Linie (Standardeinstellung)

2: 12 Pixel Strich, 4 Pixel Lücke

3: 3 Pixel Strich, 5 Pixel Lücke

4: 7 Pixel Strich, 3 Lücke, 3 Strich, 3 Lücke

5: 8 Pixel Strich, 8 Pixel Lücke

6: 4 Pixel Strich, 3 Lücke, 2 Strich, 2 Lücke, 2 Strich, 3 Lücke

7: durch LINE PATTERN frei definirbar

Ist durch LINE WIDTH eine andere Linienbreite als 1 gegeben, so wird immer eine durchgezogene Linie gezeichnet.

HINWEIS: Um die Lücke zu zeichnen wird immer Farbe 0 verwendet. Deshalb ist es in einer Schwarz-Weiß-Auflösung nicht ohne weiteres möglich eine weiße gestrichelte Linie auf schwarzem Grund zu zeichnen. Man muß zusätzlich den geeigneten Zeichenmodus (MODE =) wählen.

LINE WIDTH

Typ: Befehl

Syntax: LINE WIDTH = <num.Ausdruck>

LINE WIDTH = <Linienbreite in Pixeln>

Erklärung: Stellt die Linienstärke in Pixeln ein für alle DRAW- Befehle und für alle umrahmten gefüllten Flächen ein (siehe PBOX, BOX usw.). Die aktuelle VDI-Version unterstützt nur ungerade Linienstärken.

LIST

Typ: Befehl

Syntax: LIST [<Marke>][{-|,}[<Marke>]]

Erklärung: Gibt ein Programm-Listing auf dem Bildschirm aus. Ohne Parameter wird das gesammte Listing ausgegeben. Ist nur der erste Parameter genannt, wird die entsprechende Programmzeile gelistet. Ist der Marke ein "-" oder "," vorgestellt bzw. angehängt, so wird ein Listing vom Anfang bis zur angegebenen Zeilennummer, bzw. von der angegebenen Zeilennummer bis zum Ende ausgegeben.

Sind zwei Marken durch "-" oder "," getrennt gegeben, so wird der entsprechende Programmabschnitt aufgelistet.

Achtung: Steuerzeichen im Programmtext werden bei LIST mit ausgegeben. Beim Drucken oder Schreiben in eine Datei mit CMD sollte daher LLIST verwendet werden (LLIST gibt keine Steuerzeichen aus)

Beispiel:
```
LIST 10-30
LIST Schleife,
```

LIST$

Typ: Dieses Wort ist reserviert. Bitte nicht verwenden.

LLIST

Typ: Befehl

Syntax: LLIST [<Marke>][{-|,}[<Marke>]]

Erklärung: Gibt ein Programm-Listing auf dem Drucker aus. Ohne Parameter wird das gesammte Listing ausgegeben. Ist nur der erste Parameter genannt, wird die entsprechende Programmzeile gelistet. Ist der Marke ein "-" oder "," vorgestellt bzw. angehängt, so wird ein Listing vom Anfang bis zur angegebenen Zeilennummer, bzw. von der angegebenen Zeilennummer bis zum Ende ausgegeben.

Sind zwei Marken durch "-" oder "," getrennt gegeben, so wird der entsprechende Programmabschnitt aufgelistet.

Steuerzeichen im Programmtext werden nicht mit ausgegeben. Siehe hierzu auch CMD und LIST.

Beispiel:
```
LLIST 10-30
LLIST Schleife,
```

LN

Typ: Funktion

Syntax: LN (<num.Ausdruck>)

Erklärung: Berechnet den natürlichen Logarithmus des numerischen Ausdrucks. Der Definitionsbereich umfaßt alle positiven reellen Zahlen. Negative Argumente führen zu "Illegal function call".

Beispiel:

```
0 PRINT LN(10), LN(10#)
1 PRINT LN(-1)

 2.3025851       2.302585092994046
? Illegal function call in 1
```

LOAD

Typ: Befehl

Syntax: LOAD <Sringausdruck>

LOAD <Dateiname>

Erklärung: Die genannte Datei wird als Basic-Programm geladen.

Alle Variableninhalte und Dimensionierungen werden gelöscht, offene Dateien werden geschlossen, Multi-Tasking-Aufrufe werden abgeschaltet. COMMON-Anweisung werden gelöscht.

Es ist unerheblich, ob das Programm als OMIKRON.BASIC-Programm oder ASCII-Datei gespeichert wurde. Das Programm muß im ASCII-Format Zeilennummern haben. Doppelte Zeilen können nicht geladen werden. Siehe auch SAVE.

LOC

Typ: Funktion

Syntax: LOC(<Dateinummer>)

Erklärung: Gibt die Nummer des zuletzt gelesenen oder geschriebenen Datensatzes in einer Random-Access Datei an. Die Datei muß zuvor durch OPEN "R" geöffnet worden sein. Hat noch kein Zugriff auf die Datei stattgefunden, oder wurde über das Dateiende hinaus gelesen, so ergibt die Funktion den Wert 0.

LOCAL

Typ: Befehl

Syntax: LOCAL <Variable>[=<Ausdruck>][[,<Variable>[=<Ausdruck>]]]

Erklärung: Definiert eine oder mehrere Variablen in einer Prozedur oder mehrzeiligen Funktion als lokal und weist ihnen gegebenenfalls den genannten Ausdruck zu.

Die lokale Variable hat nur innerhalb dieser Routine Gültigkeit. Ruft sich die Routine selbst auf (rekursiv), so unterscheiden sich lokale Variablen der einzelnen Aufruftiefen. LOCAL kann nicht in GOSUB-Unterroutinen oder in Schleifen verwendet werden!

Beispiel:

```
 0 A=15
 1 Beispiel
 2 PRINT A,B
 3 END
 4 DEF PROC Beispiel
 5   LOCAL A=16
 6    B=10
 7    PRINT A,B
 8 END_PROC

16          10
15          10
```

LOCATE

Typ: Befehl

Syntax: LOCATE <num.Ausdruck>,<num.Ausdruck>

LOCATE <Zeile>,<Spalte>

Erklärung: Positioniert den Cursor auf die angegebene Bildschirmstelle. Die oberste Zeile hat hierbei die Nummer eins, während die Spaltenzählung bei 0 beginnt.

Siehe auch @, CSRLIN, POS.

Beispiel:

```
LOCATE 3,5: PRINT "Das ist ein Test"

          Das ist ein Test
```

LOCK

Typ: Befehl

Syntax: LOCK {<String-Ausdruck>|OFF}

Erklärung: Wenn Sie mittels [CONTROL] [D] einen Teil Ihres Programms eingeklappt haben, können Sie diesen Teil mit einem Passwort vor dem Ausklappen schützen. Bitte beachten Sie, daß <String-Ausruck> immer genau 6 Zeichen lang sein muß (z.B. "CODE01". Gehen Sie dabei folgendermaßen vor:

Geben Sie LOCK und ihr Passwort ein. Klappen Sie jetzt alle Teile ein, die gegen Auflisten geschützt werden sollen. Mit LOCK OFF schalten Sie jetzt das letzte eingegebene Passwort wieder aus, danach können Sie jetzt geschützte und eingeklappte Programmteile nicht mehr ausklappen.

Das Passwort und der Zustand (eingeklappt) werden mit dem Programm gesichert. Sie können also Ihre Programm ohne Probleme als Quelltext weitergeben und einige Teile davon einklappen und über LOCK schützen. ACHTUNG: Merken Sie sich Ihr Passwort gut, da Sie sonst die eingeklappten Teile nicht wieder ausklappen können.

LOF

Typ: Funktion

Syntax: LOF(<num.Ausdruck>)

LOF(<Dateinummer>)

Erklärung: Gibt die Länge einer sequentiellen oder "U"-Datei in Zeichen, einer Random-Access-Datei in Datensätzen an. Ist die sequentielle Datei zum Schreiben geöffnet, werden nur die neu geschriebenen Daten gezählt. Ist sie zum Lesen geöffnet wird das Dateiende-Zeichen (CHR$(26)) mitgezählt. Die Datei muß zuvor mit OPEN geöffnet worden sein.

LOG

Typ: Funktion

Syntax: LOG(<num.Ausdruck>,<num.Ausdruck>)

LOG(<Basis>,<Wert>)

Erklärung: Errechnet den Logarithmus des Wertes zur angegebenen Basis. Basis und Wert müssen beide größer 0 sein.

Beispiel:
```
PRINT LOG(2,1024), LOG(3,81), LOG(10#,8475)

 10          4          3.9281397068751198
```

LOW

Typ: Funktion

Syntax: LOW(<num.Ausdruck>)

Erklärung: Ermittelt die unteren 16 Bit des in Integer-Langwort gewandelten numerischen Ausdrucks. Gegenstück zu dieser Funktion ist HIGH.

Anmerkung: LOW(X) <> X AND $FFFF!

Beispiel:
```
PRINT HEX$(LOW($12345678)), LOW(1025), LOW($1234FFFE)

 5678          1          -2
```

LOWER$

Typ: Funktion

Syntax: LOWER$(<Stringausdruck>)

Erklärung: Wandelt alle im Stringausdruck enthaltenen Buchstaben in kleine Buchstaben um. Auch die Umlaute Ä, Ö, Ü werden umgewandelt.

Gegenstück zu dieser Funktion ist UPPER$.

Beispiel:
```
PRINT LOWER$("Dies ist ein Test: ÜÖÄ")
dies ist ein test: üöä
```

LPEEK

Typ: Funktion

Syntax: LPEEK(<num.Ausdruck>)

Erklärung: Liest an der durch numerischen Ausdruck gegebenen Adresse ein Langwort. Der numerische Ausdruck muß eine gerade Zahl sein, da sonst ein Bus-Fehler auftritt.

Siehe auch PEEK, WPEEK, POKE, WPOKE, LPOKE.

Beispiel:
```
PRINT "Basisadresse des Betriebssystems:";
                                HEX$(LPEEK($4F2))

Basisadresse des Betriebssystems: FC0000
```

LPOKE

Typ: Befehl

Syntax: LPOKE <Addresse>,<Wert>

Erklärung: Legt Wert an der gegebenen Adresse als ein Langwort ab. Der numerische Ausdruck muß gerade sein, dasonst ein Bus-Fehler auftritt.Der Wert muß zwischen -2147483648 und 2147483647 liegen.Siehe auch POKE, WPOKE, PEEK, WPEEK, LPEEK.

LPOS

Typ: Funktion

Syntax: LPOS(<num.Ausdruck>)

Erklärung: Ermittelt die Anzahl der seit dem letzten Line-Feed (CHR$(10)) an den Drucker übergebenen Zeichen. Sind darin keine Steuerzeichen enthalten, und befindet sich der Drucker nicht im Grafikmodus, entspricht dies der Druckspalte.Der numerische Ausdruck ist ohne Bedeutung, muß aber angegeben werden!

LPRINT

Typ: Befehl

Syntax: LPRINT [USING <Stringausdruck>] [[{;|,}<Ausdruck>]][{,|;}]

Erklärung: Genau wie bei PRINT werden die folgenden Ausdrücke ausgegeben allerdings auf den Drucker. Für die Formatierung und Tabulierung der Ausgabe gelten die gleichen Regeln wie für PRINT. Auch die zusätzliche USING Anweisung läßt sich genau wie bei PRINT USING einsetzen. Hinweis: Wenn mittels MODE LPRINT "D" auf Ländermodus deutsch umgeschaltet wurde, werden alle deutschen Umlaute und "ß" auf Epson-kompatiblen Druckern automatisch richtig ausgegeben. Diese Zeichenkonvertierung kann sich unter Umständen störend auswirken, wenn Steuerzeichen oder eine Grafik zum Drucker

übertragen werden soll. Man sollte also zuvor stets mit MODE LPRINT "USA" die Zeichenkonvertierung abschalten.

Beispiel:

```
Ø MODE LPRINT "USA"
1 LPRINT "Ä Ö Ü ü ö ä ß"
2 MODE LPRINT "D"
3 LPRINT "Ä Ö Ü ü ö ä ß"
```

LSET

Typ: Befehl

Syntax: LSET <String-Variable>=<Stringausdruck>

Erklärung: Der Stringausdruck wird linksbündig in <String- Variable> eingesetzt, ohne daß deren Länge verändert wird. Hierzu wird der Stringausdruck hinten abgeschnitten oder durch Leerzeichen ergänzt. Dieser Befehl wird besonders bei Zuweisungen an Puffervariable (siehe FIELD) benutzt.

Siehe auch RSET.

Beispiel:

```
0 A$= SPACE$(20)
1 LSET A$="OMIKRON"
2 PRINT "*";A$;"*"

*OMIKRON             *
```

MAT

Typ: Befehl

Syntax: MAT <Feldvariable>{+|-|*|/}<Feldvariable>

MAT <Feldvariable>=<Feldvariable>[{+|-|*|/}<Feldvariable>]

1: MAT <Feldvariable>=1

2: MAT <Feldvariable>=<Feldvariable>

3: MAT <Feldvariable>{+|-|*|/}<Feldvariable>

4: MAT <Feldvariable>=<Feldvariable>{+|-|*|/}<Feldvariable>

Erklärung: Zunächst ein paar grundsätzliche Dinge: Matrixbefehle funktionieren nur mit zweidimensionalen Fließkommafeldern. Dabei spielt es keine Rolle, ob Sie einfache oder doppelte Genauigkeit verwenden. Die Größe der Matrizen wird durch die angegebenen Indices bestimmt. Die Zählung der Elemente beginnt wie immer bei Null. "Matrix!(2,2)" meint also eine dreireihige quadratische Matrix. "Matrix!(N,M)" ist entsprechend eine N+1,M+1 Matrix. Wie immer bei der Verwendung von Feldern kann der Index auch weggelassen werden, was genauso behandelt wird, als hätte man den höchsten Index angegeben. Wurde das Feld "Matrix!" also auf 3,3 dimensioniert dann ist "Matrix!(,)" eine vierreihige quadratische Matrix.

Bei der Verwendung des Befehls MAT sind vier verschiedene Syntaxen zu unterscheiden:

1. Erzeugen einer Einheitsmatrix. Der quadratischen Matrix wird die Einheitsmatrix zugewiesen.

2. Die Matrixzuweisung: Es werden alle Elemente der rechten Seite der linken zugewiesen.

   Hinweis: dieser Befehl funktioniert mit beliebigen Felder, die nicht die speziellen Anforderungen von Matrizen im Sinne von OMIKRON.Basic genügen. Es wird einfach alles von rechts nach links kopiert bis zum angegebenen Element. Eine Typkonvertierung ist dabei natürlich nicht möglich.

3. Skalare Matrix-Operationen: Je nach Operator wird:
   - zu jedem Element ein Wert addiert
   - von jedem Element ein Wert subtrahiert
   - jedes Element mit einem Wert multipliziert
   - jedes Element durch einen Wert dividiert

4. Vektorielle Matrix-Operationen: Je nach Operator werden:
   - zwei Matrizen addiert (elementweise)
   - zwei Matrizen subtrahiert (elementweise)
   - zwei Matrizen multipliziert
   - zwei Matrizen dividiert (mit der Inversen multipliziert)

Bei der Matrizen-Multiplikation bzw. Division ist auf eine geeignete Wahl der Matrix-Dimensionen zu achten. Es gilt:

A(P,Q)=B(P,N)*C(N,Q). Bei der Division muß zusätzlich die Matrix C quadratisch sein.

Beispiel:

```
0 N=3
1 DIM Matrix!(N,N),Mat_A!(N,N),Mat_B!(N,N),Mat_C!(N,N)
2 DIM Array(10),Feld(10)
3 '
4 MAT Matrix!(,)=1
5 MAT Mat_A!(1,1)+2
6 MAT Mat_C!(,)=Mat_B!(,)+Mat_A!(,)
7 MAT Matrix!(1,2)=Mat_A!(1,2)
8 MAT Feld(5)=Array(6)' der kleinere Index gilt
```

MAT CLEAR

Typ: Befehl

Syntax: MAT CLEAR <Feldvariable>

Erklärung: MAT CLEAR löscht ein beliebiges Feld bis zum angegebenen maximalen Index. Wird kein Index angegeben, so wird das gesamte Feld gelöscht.

Beispiel:

```
0 N=3
1 DIM Matrix!(N,N)
2 DIM Array(10)
3 '
4 MAT CLEAR Matrix!(,)
5 MAT CLEAR Array(5)
```

MAT INV

Typ: Befehl

Syntax: MAT <Feldvariable>= INV <Feldvariable>

Erklärung: Berechnet die inverse Matrix. Beide Matrizen müssen quadratisch sein. Existiert die inverse Matrix nicht (Determinate = 0), so wird eine Fehlermeldung erzeugt.

Beispiel:

```
0 N=3
1 DIM Mat_A!(N,N),Mat_B!(N,N)
2 '
3 MAT Mat_A!(,)=1
4 MAT Mat_B!(,)= INV Mat_A!(,)
```

MAX

Typ: Funktion

Syntax: MAX (<Ausdruck>,<Ausdruck>)

Erklärung: Gibt den größeren der beiden Ausdrücke als Funktionswert.

Vergleiche zwischen numerischen und Stringausdrücken sind unzulässig. Siehe auch MIN.

Beispiel:

```
PRINT MAX(1,4), MAX(-6, PI), MAX("a","B")

 4          3.1415926535897932       a
```

MEMORY

Typ: Funktion

Syntax: MEMORY(<num.Ausdruck>)

MEMORY(<Blocklänge|-1>)

Erklärung: Die MEMORY-Funktion dient zum reservieren von GEMDOS-Speicher. Sie entspricht weitgehend der zugehörigen Betriebssystemfunktion MALLOC. Mit MEMORY reservierte Speicherblöcke werden jedoch bei Programmende oder bei CLEAR automatisch freigegeben. Der Aufruf mit minus eins als Parameter liefert die Länge des größte freien Blocks zurück. Ansonsten wird die Speicheradresse zurückgegeben, ab der der verlangte Speicherblock zur Verfügung steht. Um ausreichend große Mengen GEMDOS-Speicher mit MEMORY reservieren zu können, muß der entsprechende Parameter bei CLEAR vergrößert werden. Steht nicht ausreichend Speicher zur Verfügung, wird ein "Out of memory" ausgelöst.

Um Speicherblöcke, die Sie per MEMORY angelegt haben, wieder freizugeben verwenden Sie FRE.

Achtung: Pro MEMORY-Block benötigt das BASIC intern 4 Bytes. Außerdem sollten Sie darauf achten, daß MALLOC nicht öfter als 100 mal ausgeführt wird.

Beispiel:
```
Speicher=MEMORY(1024)
```

MEMORY_BLOCK

Typ: Befehl

Syntax: MEMORY_BLOCK <Zwei Ziffern> , <Größe> , <Adressrückgabevariable>

Erklärung: Das MEMORRY_BLOCK-Kommando erzeugt schon beim Eingeben in Ihr Programm einen Speicherblock der angegebenen Größe, der in Ihrem Programm fortan erhalten bleibt. Dieser reservierte Speicherblock (=Memoryblock, engl.) wird in Ihrem Programm mit abgespeichert und eingeladen. Dieser Speicherblock wird identifizeirt durch die beiden Ziffern direkt hinter dem MEMORY_BLOCK Befehl. Wird ein weiterer MEMORY_BLOCK mit ***gleichen*** zwei Ziffern erzeugt, der eine geringere Größe als der erste hat, so wird der erste dabei zerstört.

Da der Speicherblock also durch die zwei Ziffern gekennzeichnet ist, sind maximal 100 Speicherblöcke in einem Programm möglich. ***Achtung: Es müssen immmer GENAU zwei Ziffern sein (z.B. 03, 00 oder 23).*** die Größe des Blocks muß eine gerade Zahl sein.

Wenn Ihr Programm über den MEMORY_BLOCK-Befehl läuft (nach RUN), wird die Adresse des Speicherblocks in die Rückgabevairable geschrieben. Erst, wenn Sie Ihr Programm mit RUN starten, wird die absolute Adresse des Speicherblocks festgelegt (Änderungen im Programm veränder z.B. die Adresse). Maschinenspracheprogramme in durch MEMORY_BLOCK angelegten Speicherbereichen sollten also keine absoluten Adressen verwenden!

Beispiel:

```
MEMORY_BLOCK 00,32000,Titelbild
COMPILER OFF
BLOAD "TITEL.PIC",Titelbild
COMPILER ON
PRINT CHR$(27);"f";
MEMORY_MOVE Titelbild,32000 TO LPEEK($44E)
WAIT 3
```

Wenn Sie dieses Programm im Interpreter starten, wird das Bild "TITEL.PIC" in den Speicherblock geladen. Sobald das Programm compiliert wird, ist das Titelbild Bestandteil des Programms und das Programm muß es nicht ständig nachladen.

MEMORY_MOVE[B]

Typ: Befehl

Syntax: MEMORY_MOVE[B] <Quelladresse>, <Länge> TO <Zieladresse>

Erklärung: Das Bewegen von Speicherbereichen kann man am besten mit dem MEMORY_MOVE-Befehl bewerkstelligen. Das Speicherstück, das bei <Quelladresse> beginnt und <Länge> lang ist, wird an die <Zieladresse> kopiert. Bei MEMORY_MOVE müssen alle Parameter gerade Zahlen sein, während MEMORY_MOVEB auch ungerade Zahlen akzeptiert, dafür aber langsamer ist.

Beispiel:

```
Scr_Addr= LPEEK($44E)
Save_Buf= MEMORY(32000)
MEMORY_MOVE Scr_Addr,32000 TO Save_Buf ' rettet Bild-
schirm
FILESELECT(Path$),Name$,Button ' "zerstört" Bildschirm
MEMORY_MOVE Save_Buf,32000 TO Scr_Addr ' restauriert
Bildschirm
```

MERGE

Typ: Befehl

Syntax: MERGE <Stringausdruck>

MERGE <Programmname>

Erklärung: Lädt zum aktuellen Basic-Programm das in Stringausdruck genannte hinzu. Das zu ladende Programm muß als ASCII-Dastei vorliegen (als Block mit Zeilennummern gesichert oder SAVE ,A).

Bei gleichen Zeilennummern, wird die alte Zeile gelöscht (siehe auch RENUM).

MID$

Typ: Befehl

Syntax: MID$(<String-Variable>,<num.Ausdruck>,<num. Ausdruck>)=<Stringausdruck>

MID$(<String-Variable>,<Stelle>,<Anzahl>)=<Stringausdruck>

Erklärung: Weist der String-Variable ab der angegebenen Stelle den Stringausdruck zu, max. die angegebene Anzahl Zeichen. Dabei wird die Länge der String-Variablen nicht verändert.

Einen String mit Zeichen auffüllen kann man mit MID$ nicht!

Beispiel:

```
Ø Firma$="OMIKRON Soft + Hardware GmbH"
1 MID$(Firma$,8,1)="."
2 PRINT Firma$

OMIKRON.Soft + Hardware GmbH
```

MID$

Typ: Funktion

Syntax: MID$(<Stringausddruck,<num.Ausdruck>[,<num.Ausdruck>])

MID$(<Stringausdruck>,<Stelle>[,<Länge>])

Erklärung: Ergibt einen Teilstring des Stringausdrucks beginnend ab der gegebenen Stelle mit der gegebenen Länge. Ist Länge nicht gegeben oder größer als der verbleibende Teil des Stringausdrucks, so endet der Teilstring am Ende des Stringausdrucks.

Beispiel:

```
Ø A$="OMIKRON.Software"
1 PRINT MID$(A$,6,7)
2 PRINT MID$("abcdefg",4)

ON.Soft
defg
```

MIN

Typ: Funktion

Syntax: MIN (<Ausdruck>,<Ausdruck>)

Erklärung: Gibt den kleineren der beiden Ausdrücke als Funktionswert.

Vergleiche zwischen numerischen und Stringausdrücken sind unzulässig.

Siehe auch MAX.

Beispiel:
```
PRINT MIN(-6,0), MIN(7,7.1), MIN("a","b")

-6        7         a
```

MIRROR$

Typ: Funktion

Syntax: MIRROR$(<Stringausdruck>)

Erklärung: Spiegelt den gegebenen Stringausdruck. Das erste Zeichen wird mit dem letzten vertauscht, das zweite mit dem vorletzten etc.

Beispiel:
```
0 PRINT MIRROR$("12340")
04321
```

MKD$

Typ: Funktion

Syntax: MKD$(<num.Ausdruck>)

Erklärung: Wandelt den numerischen Ausdruck in eine doppelt-genaue-Fließkommazahl um, und diese in einen 10 Zeichen langen Stringausdrucks. Umkehrfunktion zu CVD.

Beispiel:
```
0 Zahl$=MKD$( PI )
1 FOR I=1 TO 10
2   PRINT HEX$(ASC(MID$(Zahl$,I,1)));
3 NEXT I

 0 4 C9 F DA A2 21 68 C2 36
```

MKDIR

Typ: Befehl

Syntax: MKDIR <Stringausdruck>

Erklärung: Legt einen neuen Ordner an. Der Ordnername wird in String-Ausdruck mit oder ohne Pfad angegeben, in letzterem Fall wird der Standardpfad benutzt (siehe CHDIR).

MKI$

Typ: Funktion

Syntax: MKI$(<num.Ausdruck>)

Erklärung: Wandelt den numerischen Ausdruck in eine Integer-Wort-Zahl um, und diese in einen 2 Zeichen langen Stringausdruck. Umkehrfunktion zu CVI.

Beispiel:

```
PRINT MKI$($5445)+MKI$($5354)
TEST
```

MKIL$

Typ: Funktion

Syntax: MKIL$(<num.Ausdruck>)

Erklärung: Wandelt den numerischen Ausdruck in eine Integer-Langwort-Zahl um, und diese in einen 4 Zeichen langen Stringausdruck. Umkehrfunktion zu CVIL.

Beispiel:

```
PRINT MKIL$($57656C74)

Welt
```

MKS$

Typ: Funktion

Syntax: MKS$(<num.Ausdruck>)

Erklärung: Wandelt den numerischen Ausdruck in eine einfach-genaue-Fließkommazahl um, und diese in einen 6 Zeichen langen Stringausdruck. Umkehrfunktion zu CVS.

Beispiel:

```
0 Zahl$=MKS$( PI )
1 FOR I=1 TO 6
2   PRINT HEX$(ASC(MID$(Zahl$,I,1)));
3 NEXT I

 0 4 C9 F DA A2
```

MOD

Typ: Operator

Syntax: <num.Ausdruck> MOD <num.Ausdruck>

Erklärung: Ermittelt den Restwert einer Division des ersten numerischen Ausdrucks durch den zweiten. Der Teiler muß natürlich ungleich 0 sein. Das Vorzeichen entspricht dem des ersten Ausdrucks.

Beispiel:

```
PRINT 10 MOD 3,-20 MOD 7,15 MOD -6

 1   -6        3
```

MODE=

Typ: Befehl

Syntax: MODE = <num.Ausdruck>

MODE = <Zeichenmodus>

Erklärung: Stellt den Zeichenmodus für alle Graphikbefehle ein.

Mögliche Einstellungen sind:

1 deckend 3 XOR

2 transparent 4 revers transparent

MODE

Typ: Befehl

Syntax: MODE <Stringausdruck>

MODE <Länderkennung>

Erklärung: Stellt den länderspezifischen Modus ein. Mögliche Einstellungen sind "D" für Deutschland, "GB" für England, "I" für "Italien" und "USA" für die Vereinigten Staaten von Amerika. Eine Besonderheit hat es mit "F" bzw. "F^" für Frankreich auf sich. Bei "F" werden die französischen

Sonderzeichen folgendermaßen erzeugt: Eine der Tasten "'", "^", "~" usw. wird eingegeben und danach die Taste des Buchstabens. Stellt man hingegen den Modus "F^" ein, muß für das "^" Zeichen das Zeichen "^" (ASCII-Code 222) eingegeben werden. Da dieses Zeichen auf der deutschen Tastatur fehlt, wurde der zweite Modus eingeführt, bei dem es reicht, die "^" Taste zu drücken. Die Ländereinstellung hat Einfluß auf Datums- und Zeitformat und auf Accents im Editor

Beispiel:

```
Ø MODE "D"
1 PRINT DATE$
2 MODE "USA"
3 PRINT DATE$
```

MODE LPRINT

Typ: Befehl

Syntax: MODE LPRINT <Stringausdruck>

MODE LPRINT <Länderkennung>

Erklärung: Die Länderkennung "D" bewirkt die richtige Ausgabe alle deutschen Umlaute und des "ß" auf einem Epson-kompatiblen Drucker. Wird als Länderkennung "USA" oder "GB" angegeben, so unterbleibt jede Zeichenumwandlung. Wichtig: die Zeichenumwandlung des MODE LPRINT Befehls wirkt nur auf die Druckerausgabe mit LPRINT (nicht etwa über OPEN "P"). Wenn mittels LPRINT Steuerzeichen zum Drucker übermittelt werden sollten oder eine Grafik ausgedruckt werden soll, so muß unbedingt MODE LPRINT "USA" eingestellt sein. Wenn dies versäumt wird, können unter Umständen Teile der Steuerzeichen oder der Grafik durch die Zeichenumwandlung verändert werden und erzielen so nicht das gewünschte Ergebnis.

Beispiel:

```
Ø MODE LPRINT "USA"
1 LPRINT "Ä Ö Ü ü ö ä ß"
2 MODE LPRINT "D"
3 LPRINT "Ä Ö Ü ü ö ä ß"
```

MOUSEBUT

Typ: Funktion

Syntax: MOUSEBUT

Erklärung: Gibt den Zustand der Maustasten an:

0: keine Taste

1: linke Taste

2: rechte Taste

3: beide Tasten (3=1+2)

Siehe auch MOUSEX, MOUSEY.

Beispiel:

```
0 PRINT CHR$(27);"fTaste bricht ab ...":PRINT
1 REPEAT
2   PRINT CHR$(27);"A";MOUSEBUT;MOUSEX;MOUSEY;
                                        CHR$(27);"K"
3 UNTIL LEN(INKEY$)
```

MOUSEOFF

Typ: Befehl

Syntax: MOUSEOFF

Erklärung: Schaltet die Maus ab (wird nicht mehr angezeigt).

Da das Betriebssystem die Anzahl der Mauseinschaltungen und Ausschaltungen mitzählt, ist darauf zu achten, daß innerhalb eines Programms die Maus genausooft wieder ein- wie ausgeschaltet wird.

Im Direktmodus sorgt OMIKRON.BASIC selbständig für korrekte Mausaktivierung.

MOUSEON

Typ: Befehl

Syntax: MOUSEON

Erklärung: Schaltet die Maus an (Maus wird angezeigt).

Siehe MOUSEOFF.

MOUSEX

Typ: Funktion

Syntax: MOUSEX

Erklärung: Ergibt die X-Koordinate der Mausposition. Dabei ist unerheblich, ob die Maus geradde angezeigt wird oder nicht.

Siehe auch MOUSEY, MOUSEBUT.

Beispiel:

```
0 PRINT CHR$(27);"fTaste bricht ab ...":PRINT
1 REPEAT
```

```
2   PRINT CHR$(27);"A";MOUSEBUT;MOUSEX;MOUSEY;
                                        CHR$(27);"K"
3 UNTIL LEN(INKEY$)
```

MOUSEY

Typ: Funktion

Syntax: MOUSEY

Erklärung: Ergibt die Y-Koordinate der Mausposition. Dabei ist unerheblich, ob die Maus geradde angezeigt wird oder nicht.

Siehe auch MOUSEY, MOUSEBUT.

Beispiel:

```
0 PRINT CHR$(27);"fTaste bricht ab ...":PRINT
1 REPEAT
2   PRINT CHR$(27);"A";MOUSEBUT;MOUSEX;MOUSEY;
                                        CHR$(27);"K"
3 UNTIL LEN(INKEY$)
```

NAME ... AS

Typ: Befehl

Syntax: NAME <Stringausdruck> AS <Stringausdruck>

NAME <Dateiname/alt> AS <Dateiname/neu>

Erklärung: a) Die genannte Datei erhält einen neuen Namen: Die Datei kann mit Pfad angegeben werden, dann muß dieser Pfad aber auch im neuen Dateinamen angegeben sein.

b) Der Pfadname ändert sich: Die Datei wird auf dem gleichen logischen Laufwerk verschoben. Ab TOS 1.4 können auch ganze Ordner verschoben werden.

NAND

Typ: Operator

Syntax: <num.Ausdruck> NAND <num.Ausdruck>

Erklärung: Verknüpft die beiden numerischen Ausdrücke "logisch nicht-und".

Beispiel:

```
0 PRINT BIN$((%1010 NAND %1100)+%10000)
```

NEW

Typ: Befehl

Syntax: NEW [<Stringausdruck>]

NEW [<Dateiname>]

Erklärung: Löscht das im Speicher befindliche Basic-Programm, alle Variableninhalte und Dimensionierungen. Offene Dateien werden geschlossen, ON ERROR GOTO- und ON TIMER GOSUB-Aufrufe werden abgeschaltet. COMMON-Anweisung werden gelöscht. Optional kann ein Name des neuen Programms, das anschließend geschrieben wird, angegeben werden.

NEXT

Typ: Befehl

Syntax: NEXT [<num. Variable>][[,<num. Variable>]]

NEXT [<Schleifenvariable>][[,<Schleifenvariable>]]

Erklärung: Beendet eine FOR-NEXT-Schleife. Ist keine Schleifenvariable angegeben, so bezieht sich das NEXT immer auf das letzte FOR. Durch die Aufzählung mehrerer Schleifenvariablen können mehrere Schleifen auf einmal beendet werden.

Siehe auch FOR ... TO ... STEP ... NEXT.

NDC

Typ: Befehl

Syntax: NDC [<X>,<Y>,<Breite>,<Höhe>]

NDC [<X1>,<Y1> TO <X2>,<Y2>]

Erklärung: Nach dem NDC-Befehl funktionieren alle Grafik- Befehle so, als wenn Sie einen Bildschirm hätten, der bei den Koordinaten 0,0 unten links startet und bis zu den Koordinaten 32767, 32767 oben rechts geht. Die Koordinaten werden bei den jeweiligen Grafikbefehlen dann automatisch so umgerechnet, daß sie in das von Ihnen im NDC-Befehl angegebenen Rechteck passen. NDC ohne Koordinatenangabe schaltet einfach wieder auf die physikalischen Koordinaten um.

Achtung: Im NDC-Modus ist der Punkt 0,0 in der linken unteren Ecke des gegebenen Rechtecks. Normalerweise ist der Punkt 0,0 in der linken oberen Ecke des Bildschirms.

NOISE

Typ: Befehl

Syntax: NOISE <num.Ausdruck>,<num.Ausdruck>

NOISE <Nummer>,<Frequenz>

Erklärung: Schaltet Rauschen auf die Tonkanäle des Sound-Chips.

Nummer gibt die aufzuschaltenden Tonkanäle:

1: Tonkanal 1

2: Tonkanal 2

4: Tonkanal 3

Durch entsprechende Adddition können mehrere Tonkanäle angesprochen werden.

Frequenz muß ein Wert von 0 bis 31 sein. Je kleiner der Wert, desto höher ist das Rauschen.

Um das Rauschen zu hören muß den entsprechenden Kanälen mittels VOLUME eine Lautstärke zugewiesen werden.

Siehe auch TUNE.

NOR

Typ: Operator

Syntax: <num.Ausdruck> NOR <num.Ausdruck>

Erklärung: Verknüpft die beiden numerischen Ausdrücke "logisch nicht-oder".

Beispiel:

```
Ø PRINT BIN$((%1Ø1Ø NOR %11ØØ)+%1ØØØØ)

 1
```

NOT

Typ: Funktion

Syntax: NOT <num.Ausdruck>

Erklärung: Negiert den num. Ausdruck bitweise. Da alle Ausdrücke ungleich 0 als wahr angesehen werden, macht NOT nicht unbedingt aus einem wahren Ausdruck einen falschen.

Beispiel:

```
Ø Ok=1
1 PRINT NOT(2>1), NOT Ok

 Ø       -2
```

OCT$

Typ: Funktion

Syntax: OCT$(<num.Ausdruck>)

Erklärung: Wandelt den numerischen Ausdruck in eine Zeichenkette um, die den gerundeten Wert des Ausdrucks als Oktalzahl darstellt.

Beispiel:
```
PRINT OCT$(&12345678), OCT$(-16)

 2215053170     -20
```

OFF, ON

Erklärung: Siehe OUTLINE, LOCK

ON ERROR GOTO

Typ: Befehl

Syntax: ON ERROR GOTO {<Marke>|0}

Erklärung: Wenn während des Programmlaufs ein Fehler auftritt, der eine Fehlermeldung und den Programmabbruch zur Folge hätte, kann dieser mit ON ERROR GOTO abgefangen werden. Bei Auftreten des Fehlers wird zur angegebenen Marke verzweigt. Wird statt einer Marke eine Null angegeben so ist die Fehlerüberwachung wieder abgeschaltet. Die eigene Fehlerbehandlung muß unbedingt mit RESUME abgeschlossen werden, damit erkennbar ist, was zur Fehlerbehandlung und was zum normalen Programm gehört. Tritt während der Fehlerbehandlung erneut ein Fehler auf, so wird in jedem Fall abgebrochen. Wenn die Fehlerüberwachung schon innerhalb der Fehlerbehandlung abgeschaltet wird, wird der zunächst abgefangene Fehler doch noch ausgegeben und anschließend abgebrochen. Die Systemvariablen ERR, ERR$ und ERL enthalten Fehlernummer, Fehlertext und Fehlerzeile. (Siehe Fehlermeldungen). Beispiel siehe auch RESUME.

Beispiel:
```
0 ON ERROR GOTO Fehler
1 OPEN "A",1,"C:\TEST.DAT"
2   PRINT #1,"Dies ist ein Test"
3 CLOSE 1
4 END
5-Fehler
6 IF ERR =53 THEN ' falls File not found
7   OPEN "O",1,"C:\TEST.DAT"' Datei neu erzeugen
```

```
8   CLOSE 1
9   RESUME
10 ELSE
11   ON ERROR GOTO 0' andere Fehler ausgeben
12 ENDIF
```

ON ... GOSUB

Typ: Befehl

Syntax: ON <num.Ausdruck> GOSUB <Marke> [[,<Marke>]]

Erklärung: In Abhängigkeit von <num.Ausdruck> wird zu einem der durch <Marke> definierten Unterprogramm verzweigt. Ist der num. Ausdruck eins, wird zum Ersten, ist er zwei, zum Zweiten verzeigt usw. Ist der num. Ausdruck 0 oder größer als die Anzahl der angegebenen Marken, so wird überhaupt nicht verzweigt und der Befehl wird ignoriert.

Vergleiche ON GOTO.

Beispiel:

```
0 INPUT "Wert (1-3): ";Wert
1 ON Wert GOSUB Tier,Gemuese,Baum
2 END
3-Tier: PRINT "Hund": RETURN
4-Gemuese: PRINT "Kohl": RETURN
5-Baum: PRINT "Eiche": RETURN
```

ON ... GOTO

Typ: Befehl

Syntax: ON <num.Ausdruck> GOTO <Marke> [[,<Marke>]]

Erklärung: In Abhängigkeit von <num.Ausdruck> wird zu einer der angegebenen Marken verzweigt. Ist der num. Ausdruck eins, wird zur ersten, ist er zwei, zur zweiten Marke verzeigt usw. Ist der num. Ausdruck 0 oder größer als die Anzahl der angegebenen Marken, so wird überhaupt nicht verzweigt und der Befehl wird ignoriert.

Vergleiche ON GOSUB.

Beispiel:

```
0 INPUT "Wert (1-3): ";Wert
1 ON Wert GOTO Tier,Gemuese,Baum
2 END
3-Tier: PRINT "Hund": END
4-Gemuese: PRINT "Kohl": END
5-Baum: PRINT "Eiche": END
```

ON HELP GOSUB

Typ: Befehl

Syntax: ON HELP GOSUB <Marke>

Erklärung: Wenn die HELP-Taste gedrückt wird, so wird zu dem durch <Marke> definierten Unterprogramm verzweigt.

HINWEIS: Für compilerte Programme gilt: das Unterprogramm sollte keine Stringverarbeitung enthalten!

Beispiel:

```
0 ON HELP GOSUB Hilfe
1 PRINT "Bitte HELP-Taste drücken"
2 REPEAT
3 UNTIL Ende' hier steht ihr Hauptprogramm
4 END
5 '
6-Hilfe
7 Taste$= INKEY$
8 PRINT "Die Hilfe Taste wurde gedrückt"
9 Ende=-1
10 RETURN
```

ON KEY GOSUB

Typ: Befehl

Syntax: ON KEY GOSUB <Marke>

Erklärung: Immer wenn eine Taste gedrückt wird, wird zu dem durch <Marke> definierten Unterprogramm verzeigt. Wichtig: das Unterprogramm muß die gedrückte Taste sofort durch Aufrufen der INKEY Funktion abholen, da sonst erneut in das Unterprogramm verzweigt wird, was einen Stapelüberlauf und "Out of Memory" zur Folge hat.

HINWEIS: Für compilerte Programme gilt: das Unterprogramm sollte keine Stringverarbeitung enthalten!

Beispiel:

```
0 ON KEY GOSUB Taste
1 PRINT "Bitte einige Tasten drücken ESC->Ende"
2 REPEAT
3 UNTIL Ende' hier steht ihr Hauptprogramm
4 END
5 '
6-Taste
7 Taste= CVIL( INKEY$ )
```

```
8 Shiftbits=Taste SHR 24
9 Scancode= HIGH(Taste) AND $FF
10 Ascii=Taste AND $FF
11 PRINT "Shiftbits:";Shiftbits;"  Scancode:";Scancode
12 PRINT "ASCII:";Ascii
13 IF Scancode=1 THEN Ende=-1
14 RETURN
```

ON MOUSEBUT GOSUB

Typ: Befehl

Syntax: ON MOUSEBUT GOSUB {<Marke>|0}

Erklärung: Immer wenn eine der beiden Maustasten (oder beide) gedrückt sind wird in das durch <Marke> definierte Unterprogramm verzweigt. Wichtig: das Unterprogramm wird nur angesprungen, wenn sich der Zustand der Maustasten ändert: Wenn Sie also eine Maustaste drücken und dann gedrückt halten, so wird das Unterprogramm nur einmal abgearbeitet. Die Null hinter GOSUB schaltet die Mausknopfüberwachung aus.

HINWEIS: Für compilerte Programme gilt: Das Unterprogramm sollte keine Stringverarbeitung enthalten!

Beispiel:

```
0 ON MOUSEBUT GOSUB Maustaste
1 PRINT "Bitte Maustasten drücken"
2 REPEAT
3 UNTIL LEN( INKEY$ )' hier steht ihr Hauptprogramm
4 END
5 '
6-Maustaste
7 PRINT "Mausknopfzustand: "; MOUSEBUT
8 RETURN
```

ON ... RESTORE

Typ: Befehl

Syntax: ON <num.Ausdruck> RESTORE <Marke> [[,<Marke>]]

Erklärung: Der Zeiger, der auf die nächste durch READ zu lesende offene Zuweisung zeigt wird in Abhängig von <num.Ausdruck> verschoben. Ist der Wert des num. Ausdrucks eins, so gilt die erste, ist er zwei, so gilt die zweite Marke usw. Der DATA-Zeiger zeigt auf die erste durch DATA

definierte offene Zuweisung, die hinter der betreffenden <Marke> erscheint. Ist der num. Ausdruck 0 oder größer als die Anzahl der angegebenen Marken, so wird der DATA-Zeiger nicht verschoben und der Befehl ignoriert.

Beispiel:

```
0 DATA "erster Text"
1 -Tier: DATA "Hund"
2 -Gemuese: DATA "Kohl"
3 -Baum: DATA "Eiche"
4 INPUT "WERT (1-3): ";Wert
5 ON Wert RESTORE Tier,Gemuese,Baum
6 READ Text$
7 PRINT Text$
```

ON TIMER ... GOSUB

Typ: Befehl

Syntax: ON TIMER <num.Ausdruck> GOSUB {<Marke>|0}

ON TIMER <Zeit in sec> GOSUB {<Marke>|0}

Erklärung: Das durch <Marke> definierte Unterprogramm, wird alle <Zeit> Sekunden angesprungen. Wird als Sprungziel Null angegeben wird die Zeitgeberüberwachung wieder abgeschaltet. Achten Sie darauf, daß Ihr Unterprogramm nicht zuviel Zeit in Anspruch nimmt. Benötigt im Extremfall Ihr Unterprogramm mehr Zeit als das von Ihnen eingestellte Intervall, so wird es sofort nach Beenden erneut aufgerufen und es bleibt keine Zeit mehr zur Ausführung des eigentlichen Programms.

HINWEIS: Für compilerte Programme gilt: Das Unterprogramm sollte keine Stringverarbeitung enthalten!

Beispiel:

```
0 MODE "D"
1 ON TIMER .5 GOSUB Uhr
2 FOR I=1 TO 1E+7: NEXT ' hier könnte Ihr Hauptprogramm
                                                    stehen
3 END
4 '
5-Uhr
6 PRINT CHR$(27)+"j"+@(0,0)+TIME$+"
                "+DATE$+CHR$(27)+"k";
7 RETURN
```

ON TRON GOSUB

Typ: Befehl

Syntax: ON TRON GOSUB {<Marke>|0}

Erklärung: Wenn mittels TRON der Trace-Modus aktiviert wurde, wird vor jedem Befehl in das durch <Marke> gegebene Unterprogramm verzweigt. Die normale Protokollfunktion des Trace-Modus wird abgeschaltet und kann stattdessen durch ein intelligenteres Unterprogamm ersetzt werden, das nur bestimmte Dinge ausgibt oder auch Variableninhalte prüft. Hierzu haben die Systemvariablen ERR, ERR$ und ERL im TRON-Unterprogramm eine besondere Bedeutung: ERR$ liefert den nun auszuführenden Befehl als Klartext, ERL die Nummer der Zeile, die gerade ausgeführt wird. ERR enthält die interne Tokennummer, die nicht weiter von Bedeutung ist (entspricht dem Befehl in ERR$). Um sich einen Überblick zu verschaffen probieren Sie das Beispiel einmal aus. Es läßt das Programm im Einzel-Schritt-Modus ablaufen und zeigt den zugehörigen Befehl am oberen Bildschirmrand an.

Wenn statt einer Marke eine Null an gegeben wird, ist wieder die normale Protokoll-Funktion des Trace-Modus aktiviert.

HINWEIS: Im Compilat existert ein entsprechender Befehl nicht. Enthält ein Programm Befehle für Trace-Unterprogramme, werden diese vom Compiler ignoriert.

Beispiel:

```
0 ON TRON GOSUB Trace: TRON
1 REPEAT
2   PRINT I
3   I+=3
4 UNTIL I>5
5 END
6 -Trace
7 PRINT CHR$(27);"j";@(0,0);ERR$;" ";ERL;" Taste
                                    drücken ...";
8 PRINT CHR$(27);"K";CHR$(27);"k";
9 REPEAT UNTIL LEN(INKEY$): RETURN
```

OPEN

Typ: Befehl

Syntax: OPEN <Stringausdruck>,<num.Ausdruck>,<Stringausdruck>[,<num. Ausdruck>]

OPEN <Mode-Kennung>,<Dateinummer>,<Dateiname> [,<Satzlänge/ Attribute>]

Erklärung: OPEN öffnet eine Datei. Die ein Zeichen lange Mode-Kennung charakterisiert die Art der Datei, die geöffnet werden soll. OMIKRON. Basic verwaltet bis zu 16 gleichzeitig offene Dateien. Die Dateinummer muß deshalb zwischen 1 und 16 liegen und dient bei allen weiteren Datei-Operationen als Erkennungsmerkmal (Handle), welche Datei gemeint ist.

Der Dateiname enthält den Namen der Datei, wobei wieder entweder nur der Name, ein kompletter Pfad oder ein relativer Pfad verwendet werden können.

Die verschiedenen Modi im einzelnen - zunächst eine Übersicht:

"I"	"Input"	Sequentielle Datei; nur zu lesen
"O"	"Output"	Sequentielle Datei; nur zu schreiben löscht eine bereits bestehende Datei
"A"	"Append"	Sequentielle Datei; nur zu schreiben an eine bestehende Datei wird angefügt
"R"	"Random"	Random-Access-Datei; lesen und schreiben feste Satzlänge; Verwendung von FIELD/PUT/GET
"U"	"User"	Random-Access-Datei; lesen und schreiben keine feste Satzlänge; Verwendung von PUT/GET/SEEK
"F"	"Files"	Inhaltsverzeichnis eines Laufwerks oder Pfades lesen.
"P"	"Printer"	öffnet einen Kanal auf den Drucker
"V"	"V24"	öffnet einen Kanal auf die RS232- Schnittstelle.
"M"	"MIDI"	öffnet einen Kanal auf die MIDI- Schnittstelle.
"K"	"Keyboard"	ist seit der Version 3.5 nicht mehr vorhanden.

Zunächst zu den drei sequentiellen Dateimodi "Input", "Output" und "Append". Mögliche Schreib-Lese-Funktionen: PRINT #, WRITE #, INPUT #, INPUT$(,) oder LINE INPUT #. Desweiteren können EOF und LOF ermittelt werden. WICHTIG: Um eine Datei mit "Append" zu öffnen muß diese bereits bestehen, ansonsten wird ein Fehler ausgelöst.

Die Random-Access-Datei vom Typ "R" benötigt als zusätzlichen Parameter die Satzlänge. Wird keine Satzlänge angegeben so gilt die voreingestellte Satzlänge von 128. Vor jedem Schreib- oder Lesevorgang muß mittels FIELD eine oder mehrere Puffervariablen

definiert werden. GET und PUT lesen bzw. schreiben dann einen beliebigen Datensatz. Der erste Datensatz hat die Nummer 1. Wird ein Satz geschrieben, der noch nicht innerhalb der Datei existierte, so wird die Datei automatisch verlängert. LOF ermittelt die Dateilänge in Sätzen. LOC ergibt die Nummer des letzten gelesenen oder geschriebenen Satzes. EOF zeigt an, ob ein weiterer Satz zum Lesen bereitsteht.

Die "User" Datei benötigt keine Angabe der Satzlänge. Man kann beliebige Datenmengen lesen oder schreiben. Die Vereinbarung einer Puffervariablen entfällt, man schreibt/liest mit PUT/GET direkt einen Speicherblock oder einen String. Zusätzlich kann der Schreib-Lese-Zeiger mit SEEK neu positioniert werden. LOF ergibt die Dateilänge in Bytes. LOC ergibt die Position des Schreib-Lese-Zeigers in Byte (beginnend mit 0). EOF zeigt an, ob ein weiteres Byte zum Lesen bereitsteht.

Eine "Inhaltsverzeichnisdatei" dient zum Auslesen eines Inhaltsverzeichnisses eines Laufwerk oder eines Pfades. Dabei kann der Dateiname eine Jokerauswahl enthalten z.B. "C:*.BAS". Als zusätzlicher Parameter muß eine Kennzahl für die gewünschten Dateiattribute angegeben werden. Diese Kennzahl setzt sich wie folgt zusammen:

+1 für schreibgeschützte Dateien

+2 für verborgene Dateien

+4 für Systemdateien

+8 für Diskettenname

+16 für Ordner

Wenn also nur "normale" Datei gesucht werden lautet die Kennzahl null, verborgene und Systemdateien dazu: sechs. Sollen nur Ordner gesucht werden: 16. Ein Verzeichniseintrag wird immer mit GET <Dateinummer>,0 geholt, zuvor sollte man mit EOF(<Dateinummer>) sicherstellen, daß noch ein weiterer Eintrag vorhanden ist. Um den Verzeichniseintrag auswerten zu können, sollten einige Puffervariablen gemäß folgendem Schema definieren:

```
FIELD ‹Dateinummer›,21,1 AS Att$,2 AS Tim$,2 AS Dat$,4 AS
                                        Len$,14 AS Name$
```

Der Aufbau entspricht genau dem Aufbau der sogenannten DTA (Betriebssystem-Struktur). Zu beachten ist, daß Name$ in der Regel noch Null-Zeichen (CHR$(0)) enthält, die eventuell ab gespalten werden müssen (z.B. wenn Strings verkettet werden sollen).

Die Datei-Modi, welche die Schnittstellen des Rechners ansprechen funktionieren analog zu sequentiellen Datei. Es wird eben von der betreffenden Schnittstelle gelesen bzw. auf ihr ausgegeben. Besonders nützlich ist dies in Zusammenhang mit dem CMD-Befehl. Man kann so z.B. auf einfache Weise alle Bildschirmausgaben über PRINT auf den

Drucker umlenken. (siehe CMD). Funktionen wie LOF, LOC oder EOF sind bei Datei (Kanälen), die sich auf Schnittstellen beziehen nicht erlaubt.

OR

Typ: Operator

Syntax: <num.Ausdruck> OR <num.Ausdruck>

Erklärung: Verknüpft die beiden numerischen Ausdrücke logisch oder.

Beispiel:
```
Ø PRINT BIN$(%1Ø1Ø OR %11ØØ)

 1111Ø
```

OUTLINE

Typ: Befehl

Syntax: OUTLINE {ON|OFF}

Erklärung: Die Außenlinie (Umrahmung) aller gefüllten Rechtecke, Kreise usw. können Sie mit diesem Kommando an- oder ausschalten. Ein Beispiel macht dies wohl am besten deutlich:

Beispiel:
```
Ø CLS: FILL COLOR=Ø
1 FILL STYLE=2,4
2 OUTLINE OFF: PCIRCLE 1ØØ,1ØØ,8Ø
3 OUTLINE ON: PCIRCLE 2ØØ,1ØØ,8Ø
```

PALETTE

Typ: Befehl

Syntax: PALETTE [<num.Ausdruck>],[[,<num.Ausdruck>]]

PALETTE [<Farbregister 0>],[[,<Farbregister 1>]]

Erklärung: Setzt eine neue Farbpalette. Die Farbwerte werden der Reihe nach direkt in die Register des Video-Bausteins geschrieben. Ein Farbwert setzt sich zusammen aus einer dreistelligen hexadezimalen Zahl, wobei jede Hex-Ziffer die Farbsättingung von Rot, Grün und Blau angibt. $777 steht also für weiß, $700 für Rot usw.

WICHTIG: Da dieser Befehl die Videohardware des ATARI ST voraussetzt und mit anderen Grafiksystemen nicht zusammenarbeitet, ist er in höchstem Maße inkompatibel und sollte daher nicht verwendet werden.

Stattdessen sollte der entsprechende VDI-Befehl zum Einsatz kommen:

Vs_Color(<Farbnummer>,<R>,<G>,<B>).

Die Farbsättigung wird hier allerdings in 1/1000 angegeben und ist somit unabhängig von den Fähigkeiten der Videohardware.

PBOX

Typ: Befehl

Syntax: PBOX <num.Ausdruck>,<num.Ausdruck>{ TO <num.Ausdruck>,<num.Ausdruck>|,<num.Ausdruck>,<num.Ausdruck>}

PBOX <X>,<Y>{ TO <X2>,<Y2>|<Breite>,<Höhe>}

Erklärung: Zeichnet ein gefülltes Rechteck auf den Bildschirm. Dabei sind entweder zwei gegenüberliegende Ecken anzugeben, oder eine Ecke, Breite und Höhe des Rechtecks. Gefüllt wird mit der durch FILL COLOR definierten Farbe und dem durch FILL STYLE definierten Füllstil.

Die Umrandung ist unabhängig von den Einstellungen durch LINE STYLE und LINE WIDTH immer eine durchgezogene, ein Pixel breite Linie. Die Farbe entspricht der Füllfarbe. Die Umrandung kann mit OUTLINE OFF aus- bzw. OUTLINE ON eingeschaltet werden.

Ist mittels CLIP ein Bildfenster definiert, wird außerhalb dieses Bereiches nicht gezeichnet.

Siehe auch PRBOX, BOX, RBOX.

PCIRCLE

Typ: Befehl

Syntax: PCIRCLE <num.Ausdruck>,<num.Ausdruck>,<num.Ausdruck>[,<num.Ausdruck>,<num.Ausdruck>]

CIRCLE <X>,<Y>,<Radius>[,<Startwinkel>,<Endwinkel>]

Erklärung: Zeichnet um den Mittelpunkt X,Y einen gefüllten Kreis mit dem angegebenen Radius. Optional können Start- und Endwinkel in Zehntel-Graden angegeben werden. Hierbei ist Winkel=0 rechts vom Mittelpunkt, Winkel=900 oberhalb des Mittelpunkts etc.

Gefüllt wird mit der durch FILL COLOR definierten Farbe und dem durch FILL STYLE definierten Füllstil.

Die Umrandung ist unabhängig von den Einstellungen durch LINE STYLE und LINE WIDTH immer eine durchgezogene, ein Pixel breite Linie. Die Farbe entspricht der Füllfarbe. Die Umrandung kann mit OUTLINE OFF aus- bzw. OUTLINE ON eingeschaltet werden. Ist mittels CLIP ein Bildfenster definiert, wird außerhalb dieses Bereiches nicht gezeichnet. Siehe auch PELLIPSE, CIRCLE, ELLIPSE.

PEEK

Typ: Funktion

Syntax: PEEK(<num.Ausdruck>)

Erklärung: Liest an der durch numerischen Ausdruck gegebenen Adresse ein Byte.

Siehe auch WPEEK, LPEEK, POKE, WPOKE, LPOKE.

PELLIPSE

Typ: Befehl

Syntax: PELLIPSE <num.Ausdruck>,<num.Ausdruck>,<num.Ausdruck>,<num.Ausdruck>[,<num.Ausdruck>,<num.Ausdruck>]

PELLIPSE <X>,<Y>,<X-Radius>,<Y-Radius>[,<Startwinkel>,<Endwinkel>]

Erklärung: Zeichnet um den Mittelpunkt X,Y eine Ellipse mit den angegebenen Radien. Optional können Start- und Endwinkel in Zehntel-Graden angegeben werden. Hierbei ist Winkel=0 rechts vom Mittelpunkt, Winkel=900 oberhalb des Mittelpunkts etc.

Gefüllt wird mit der durch FILL COLOR definierten Farbe und dem durch FILL STYLE definierten Füllstil.

Die Umrandung ist unabhängig von den Einstellungen durch LINE STYLE und LINE WIDTH immer eine durchgezogene, ein Pixel breite Linie. Die Farbe entspricht der Füllfarbe. Die Umrandung kann mit OUTLINE OFF aus- bzw. OUTLINE ON eingeschaltet werden.

Ist mittels CLIP ein Bildfenster definiert, wird außerhalb dieses Bereiches nicht gezeichnet.

Siehe auch PCIRCLE, ELLIPSE, CIRCLE.

PI

Typ: Funktion

Syntax: PI

Erklärung: Ergibt die Kreiszahl Pi als doppelt-genaue-Fließkomma-Zahl.

Beispiel:

```
Ø PRINT PI

 3.1415926535897932
```

POINT

Typ: Funktion

Syntax: POINT(<num.Ausdruck>,<num.Ausdruck>)
POINT(<X>,<Y>)

Erklärung: POINT liefert den Farbwert eines bestimmten Punktes.

POKE

Typ: Befehl

Syntax: POKE <num.Ausdruck>,<num. Ausruck>
POKE <Addresse>,<Wert>

Erklärung: Legt Wert an der gegebenen Adresse als ein Byte ab.
Der Wert muß zwischen 0 und 255 liegen.
Siehe auch WPOKE, LPOKE, PEEK, WPEEK, LPEEK.

POLYGON

Typ: Befehl

Syntax: POLYGON <Feld>

Erklärung: Zeichnet ein Polygon (beliebiges n-Eck). Die Daten der Eckpunkte müssen in einem zweidimensionalen Variablenfeld vorliegen.

Beispiel:

```
 0 DIM Pts%(1,10)
 1 READ Anzahl_Punkte
 2 FOR I=0 TO Anzahl_Punkte
 3   READ Pts%(0,I),pts%(1,I)
 4   ' Pts%(0,I)= I-ter Koordinaten-X-Wert
 5   ' Pts%(1,I)= I-ter Koordinaten-Y-Wert
 8 NEXT I
 9 CLS: POLYGON Pts%(0,Anzahl_Punkte)
10 DATA 6, 50,75, 150,125, 200,100
11 DATA 150,74, 50,125, 50,75
```

Bitte beachten Sie: Das Variablenfeld für POLYGON muß ein Integer-Word-Feld sein, das auf (1,X), X=Anzahl der maximalen Punkte, dimensioniert wurde. Alles andere funktioniert nicht.

POS

Typ: Funktion

Syntax: POS (<num.Ausdruck>)

Erklärung: Ergibt die Spalte, in der sich der Cursor gerade befindet. Die Spaltenzählung beginnt bei 0. Der Parameter ist ohne Bedeutung.

Siehe auch CSRLIN, LOCATE, @.

Beispiel:

```
PRINT "X"*10, POS(0)

XXXXXXXXXX        16
```

PPOLYGON

Typ: Befehl

Syntax: PPOLYGON <Feld>

Erklärung: Zeichnet ein Polygon wie bei POLYGON, jedoch ausgefüllt mit einem Muster.

PRBOX

Typ: Befehl

Syntax: PRBOX <num.Ausdruck>,<num.Ausdruck>{ TO <num.Ausdruck> ,<num. Ausdruck> | ,<num.Ausdruck>,<num.Ausdruck>}

PRBOX <X>,<Y>{ TO <X2>,<Y2> |<Breite>,<Höhe>}

Erklärung: Zeichnet ein gefülltes Rechteck mit abgerundeten Ecken auf den Bildschirm. Dabei sind entweder zwei gegenüberliegende Ecken anzugeben, oder eine Ecke, Breite und Höhe des Rechtecks.

Gefüllt wird mit der durch FILL COLOR definierten Farbe und dem durch FILL STYLE definierten Füllstil.

Die Umrandung ist unabhängig von den Einstellungen durch LINE STYLE und LINE WIDTH immer eine durchgezogene, ein Pixel breite Linie. Die Farbe entspricht der Füllfarbe. Die Umrandung kann mit OUTLINE OFF aus- bzw. OUTLINE ON eingeschaltet werden.

Ist mittels CLIP ein Bildfenster definiert, wird außerhalb dieses Bereiches nicht gezeichnet.

Siehe auch PBOX, RBOX, BOX.

PRINT

Typ: Befehl

Syntax: PRINT [[<Ausdruck>] [,[[<Ausdruck>]<Trennung>]...]

<Ausdruck<: TAB(<num.Ausdruck>)
USING[[<Stringausdruck>]
<num.Ausdruck> oder <Stringausdruck>

<Trennung>: , oder ;

Erklärung: Der PRINT Befehl dient zur Ausgabe von Ergebnissen am Bildschirm. Er kann über den CMD Befehl auch auf andere Ausgabegeräte umgelenkt werden. Die Ausgabe läßt sich auf verschiedene Weise tabulieren: Trennung bzw. Enden auf:

Komma: der Cursor geht zur nächsten Tabulatorposition (8er Spalte)

Semikolon: der Cursor bleibt an der letzten Schreibpositon

Wenn am Ende der PRINT Anweisung weder Komma noch Semikolon erscheinen, wird automatisch ein Zeilenvorschub ausgegeben.

Eine weitere Tabuliermöglichkeit bietet der TAB Befehl. Der Cursor rückt immer auf die angegebene Tabulierspalte vor. Ist diese Spalte durch frühere Ausgaben bereits überschritten, so wird die Cursor-Position nicht verändert. Siehe auch LPRINT, WRITE,PRINT#

Beispiel:

```
0 PRINT 1,2,3
1 PRINT 1,2,
2 PRINT 3
3 PRINT TAB (10);1; TAB (20);2
```

PRINT#

Typ: Befehl

Syntax: PRINT # [[<Ausdruck>] [,[[<Ausdruck>]<Trennung>]...]

Siehe PRINT

Erklärung: Genauso wie PRINT dient auch PRINT # der Ausgabe. Es wird auf die durch Dateinummer bestimmte Datei ausgegeben. Siehe PRINT.

Beispiel:

```
0 OPEN "O",1,"C:\TEST.DAT"
1   PRINT #1,1,2,3
2   PRINT #1,4,5,6
3 CLOSE 1
```

PRINT @

Typ: Befehl (PRINT) + Funktion (@)

Syntax: PRINT @(<num.Ausdruck>,<num.Ausdruck>); ...

PRINT @(<Zeile>,<Spalte>); ...

Erklärung: Ausgabe und Syntax wie bei PRINT. Zusätzlich wird eine Startzeile und eine Startspalte angegeben, bei der die Ausgabe beginnt. Die Zählung für Zeile und Spalte beginnt bei Null.

Grundsätzlich liefert die @(<Zeile>,<Spalte>)-Funktion einen String, der über PRINT ausgegeben den Cursor an die entsprechende Position setzt. Diese Funktion kann daher überall da erscheinen, wo eine beliebiger anderer String hätte stehen können. Sie findet z.B. auch in Verbindung mit INPUT Verwendung!

Beispiel:

```
0 FOR I=0 TO 20
1   PRINT @(I,2*I);"*"
2 NEXT I
```

PRINT USING

Typ: Befehl

Syntax: PRINT USING <Stringausdruck> ...

PRINT USING <Formatstring> ...

Erklärung: PRINT USING dient zur formatierten Zahlenausgabe. Alle Tabuliermöglichkeiten und die gesamte Syntax wie bei PRINT. Zusätzlich kann an jeder Stelle der USING Befehl gefolgt von einem Steuerstring auftauchen. Die im Steuerstring getroffenen Formatanweisung gelten dann solange für jede numerische Ausgabe, bis ein neues USING Kommando auftaucht oder der PRINT Befehl zu Ende ist.

Formatierungsmöglichkeiten:

"#"	steht für eine Ziffer
"."	Dezimal-Punkt an dieser Stelle
","	Dezimal-Komma an dieser Stelle
"," & "."	Dezimal-Punkt wo angegeben; Tausender mit "," trennen
"." & ","	Dezimal-Komma wo angegeben; Tausender mit "." trennen
",,,"	Tausender mit , trennen; keine Nachkommastellen
"..."	Tausender mit . trennen; keine Nachkommastellen
"-"	Wenn negativ, Minuszeichen an dieser Stelle
"+"	Vorzeichen immer an dieser Stelle ausgeben (auch +)

Ausnahme zu +/-: Steht das + oder - direkt vor dem ersten #, *, . oder Komma, so wird das Vorzeichen direkt vor der ersten gültigen Stelle ausgegeben.

"*<Zeichen>" vorderes Füllzeichen festlegen. Das _ (underline) darf nicht Füllzeichen sein. Nicht benutzte Ziffernstellen (also #-Zeichen etc.) werden normalerweise als Leerzeichen ausgegeben. Mit z.B. "*0" werden diese Ziffernstellen als Nullen ausgegeben, d.h., man erreichteine Ausgabe mit führenden Nullen.

"_<Zeichen>" gibt das Zeichen aus, auch wenn es sich um ein Zeichen handelt, das normalerweise eine Bedeutung für den Formatstring hätte. "_#" gibt beispielsweise ein "#" aus; dieses "#" ist daher kein Platzhalter für eine Ziffer.

"^^^^" Exponent steht an dieser Stelle (erzwingt Exponentialdarstellung)

Alle weiteren Zeichen werden so ausgegeben, wie sie sind.

Hinweis: ***Wenn Sie für das Vorzeichen nicht extra eine Stelle zuweisen, beansprucht es 1 Ziffernstelle für sich (auch wenn die Zahl positiv ist). Die Formatmaske "###" kann daher nur zweistellige Werte fassen, da eine Stelle immer für das Vorzeichen reserviert bleibt. Wenn der auszugebende Wert nicht in die Formatmaske paßt, so wird die Maske selbst mit einem "%" am Anfang ausgegeben.***

WICHTIG: ***PRINT USING rundet nicht, die Nachkommastellen werden einfach abgeschnitten.***

Einschränkungen bei PRINT USING:

Sie dürfen nicht mehr als insgesamt 30 Ziffernplatzhalter definieren. Die Anzahl der Stellen für den Exponenten darf nicht kleiner sein als 4. Das Zeichen Underline (_) darf nicht Füllzeichen sein. Die gesamte Länge des Formatstrings darf nicht größer sein als 253 Zeichen.

Wenn Sie in einer Zeile, die einen PRINT USING oder USING-Befehl enthält, einen ?SYNTAX ERROR erhalten, dann könnte das auch an einem fehlerhaften Formatstring (etwa: zwei Vorzeichen) liegen.

Beispiel:

```
0 PRINT USING "####.##",1.5
1 PRINT USING "# # # # #",1000
2 Betrag!=13450.99
3 PRINT USING "**.########,##  DM",Betrag!
4 Betr#=123456789.01#
5 PRINT USING "*0### Millionen ### Tausend und ### DM, ## PF-",Betr#
```

```
6 PRINT USING "##.##### ````"; EXP(10)
7 PRINT USING "######  DM",FN Round!(Betrag!,0)
8 END
9 DEF FN Round!(Betrag!,Stellen)
10   LOCAL Scale!=10^Stellen
11   RETURN FIX(Betrag!*Scale!+.5)/Scale!
12 END_FN
```

PROC

Typ: Befehl

Syntax: [PROC] <Prozedur-Bezeichner>[()<Ausdruck>[[,<Ausdruck>]][)]

Erklärung: Ruft eine selbst definierte Prozedur (siehe DEF PROC) auf. Wie Sie der Syntaxbeschreibung entnehmen können, ist die Angabe des Wortes "PROC" optional, d.h. Sie können die Prozedur einfach durch Nennung ihres Namens aufrufen. Auch die Angabe der Klammern, die die Parameter einschließen, ist freigestellt. Ein Prozeduraufruf unterscheidet sich somit kaum von Standard-Befehlen.

Beispiel:

```
0 Zentriere "Das ist ein Test"
1 Home
2 END
3 DEF PROC Zentriere(Text$)
4    PRINT TAB((W_CHAR-LEN(Text$))/2);Text$;
5 END_PROC
6 DEF PROC Home ' Cursor in die linke obere Ecke setzen
7    PRINT CHR$(27);"H";
8 END_PROC
```

PUT

Typ: Befehl

Syntax: PUT <num.Ausdruck>,{<num.Ausdruck>[,<num. Ausdruck]|<Stringausdruck>,<num.Ausdruck>}

1: PUT <Dateinummer>,<Satznummer>

2: PUT <Dateinummer>,<Speicheradresse>,<Anzahl>

3: PUT <Dateinummer>,<Stringausdruck>

Erklärung: Nach Syntax 1 wird ein Datensatz in die durch die Dateinummer gegebene Datei geschrieben.

Die Datei muß zuvor mittels OPEN "R" geöffnet worden sein.

Die geschriebenen Daten setzen sich aus den in FIELD genannten Puffervariablen zusammen. Da die Länge der Puffervariablen nicht verändert werden darf, empfiehlt sich die Zuweisung mit LSET oder RSET.

Nach Syntax 2 bzw. 3 werden die angegebene Anzahl Bytes ab der Speicheradresse bzw. der gegebene Stringausdruckin die mit Dateinummer genannte Datei geschrieben.

Die Datei muß zuvor mit OPEN "U" geöffnet worden sein.

Beispiel:

```
0 OPEN "U",1,"BILD1.DAT"
1 XBIOS Adr,3 ' Adresse des Bildschirms ermitteln
2 PUT 1, Adr, 32000 'Bildschirmdaten in die Datei
                                          schreiben
3 CLOSE 1
```

RAD

Typ: Befehl

Syntax: RAD

Erklärung: Schaltet für trigonometrische Funktionen auf Berechnung nach Bogenmaß um (0 bis 360). Dies ist auch die Standardeinstellung. Siehe auch DEG.

RBOX

Typ: Befehl

Syntax: RBOX <num.Ausdruck>,<num.Ausdruck>{ TO <num.Ausdruck>,<num.Ausdruck>|,<num.Ausdruck>,<num.Ausdruck>}

RBOX <X>,<Y>{ TO <X2>,<Y2>|<Breite>,<Höhe>}

Erklärung: Zeichnet ein nicht gefülltes Rechteck mit abgerundeten Ecken auf den Bildschirm. Dabei sind entweder zwei gegenüberliegende Ecken anzugeben, oder eine Ecke, Breite und Höhe des Rechtecks. Linienfarbe, Linienstil und Linienbreite können über LINE COLOR, LINE STYLE, bzw. LINE WIDTH bestimmt werden.

Ist mittels CLIP ein Bildfenster definiert, wird außerhalb dieses Bereiches nicht gezeichnet.

Siehe auch BOX, PRBOX, PBOX.

READ

Typ: Befehl

Syntax: READ <Variable>[[,<Variable>]]

Erklärung: Der Variable wird eine durch DATA definierte offene Zuweisung übergeben. Die offenen Zuweisungen werden der Reihe nach gelesen. Der Zeiger, der auf die nächste zu lesende Zuweisung zeigt, kann jedoch mit RESTORE verschoben werden.

Die Zuweisung eines Stringausdrucks an eine numerische Variable oder umgekehrt ist unzulässig.

Beispiel:

```
0 DATA "Hund",A$,"Maus",A$
1 Inh=2
2 READ A$
3 READ B$
4 PRINT A$,B$
5 RESTORE 8
6 READ C$,A
7 PRINT C$,A
8 DATA "Katze",23*Inh

Hund     Hund
Katze     46
```

REGISTER

Typ: Dieses Wort ist reserviert. Bitte nicht verwenden!

REM

Typ: Befehl

Syntax: REM <beliebiger Text>

Erklärung: Durch REM abgetrennte Texte werden vom Basic nicht beachtet und dienen nur dem Programmierer zur Erläuterung oä.

Im Gegensatz zu ', das an jeder Stelle im Programm stehen kann (außer in durch " " eingeschlossene Teile), muß REM entweder am Zeilenanfang oder hinter einem Befehlstrenner stehen. Ein Befehlstrenner ist in der Regel ein ":", aber auch ELSE, ENDIF, THEN, WHILE, REPEAT sind Befehlstrenner. Der Rest dieser Zeile ist dann als Kommentartext gekennzeichnet. Befehle, die in einer Zeile hinter einem REM stehen, werden als Kommentar angesehen und nicht ausgeführt.

RENUM

Typ: Befehl

Syntax: RENUM [<num.Ausdruck>],[<num.Ausdruck>][,<num.Ausdruck>]

RENUM [<neue Zeilennummer>],[<Startzeile>][,<Schrittweite>]

Erklärung: Weist den Programmzeilen eine neue Zeilennummer zu. Gezählt wird, wenn nicht durch Schrittweite angegeben, in Einer-Schritten. Optional kann die Zeilennummer angegeben werden, an der die Neunummerierung beginnen soll, und die erste neue Programmzeile. Auch Sprunganweisungen (GOTO) und Unterprogrammaufrufe (GOSUB) werden entsprechend angepaßt.

REPEAT

Typ: Befehl

Syntax: REPEAT

Erklärung: Leitet die REPEAT-UNTIL Schleife ein. Alle was zwischen REPEAT und UNTIL steht wird solange ausgeführt bis die Bedingung hinter UNTIL erfüllt (<>0) ist. Da die Bedingung erst am Ende des ersten Durchlaufs geprüft wird, muß die Schleife mindestens einmal durchlaufen werden, selbst wenn die Bedingung nie erfüllt war.

Durch Angabe einer nie erfüllten Bedingung (die Null) kann die REPEAT-UNTIL Schleife auch als Endlosschleife verwendet werden, die nur mit EXIT verlassen werden kann.

Beispiel:

```
0 REPEAT
1   I+=1
2   PRINT I
3 UNTIL I>4
4 REPEAT
5   INPUT "Geben Sie eine Zahl ein:";Zahl
6   Summe+=Zahl
7   IF Zahl=0 THEN EXIT
8 UNTIL 0
9 PRINT Summe
```

RESERVED

Typ: Funktion

Syntax: RESERVED(<num.Ausdruck>)

Erklärung: RESERVED ermöglich den Zugriff auf interne Speicherbereiche des Interpreters bzw. Compilers. Insbesondere ergibt RESERVED(0) den Rückgabewert (das Register D0) des letzten Maschinenprogramms, das mit CALL aufgerufen wurde.

Ein weiteres wichtiges Flag wird über RESERVED(4) erreicht. Setzt man dieses Flag mit "POKE RESERVED(4),1", so wird im Compilat eine INPUT USING Anweisung abgebrochen. Der Rückgabewert lautet in diesem Fall -3.

Die Funktion RESERVED sollte im allgemeinen nicht verwendet werden!

Beispiel:

```
0 ON TIMER 10 GOSUB Tim
1 INPUT "Sie haben 10 sec.Sekunden Zeit:";Eingabe$
                                    USING "a0m",Ret,30,P
2 PRINT
3 IF Ret = -3 THEN PRINT "Zu lange gewartet"
                          ELSE PRINT Eingabe$
4 END
5 -Tim
6 POKE RESERVED(4),1
7 RETURN
```

RESTORE

Typ: Befehl

Syntax: RESTORE [<Marke>]

Erklärung: Der Zeiger, der auf die nächste durch READ zu lesende offene Zuweisung zeigt, wird verschoben. Er zeigt auf die erste durch DATA definierte offene Zuweisung, die hinter der <Marke> erscheint.

RESTORE alleine verschiebt den Zeiger auf die erste überhaupt im Programm erscheinende offene Zuweisung. Dort steht der Zeiger auch zu Programmbeginn.

Beispiel:

```
0 DATA "Katze","Maus"
1 DATA "Hund","Maus"
2-Bleistift
3 DATA "Radiergummi"
4 READ A$: PRINT A$,
5 RESTORE 5-4  'Sprungziel berechnet, siehe GOTO
6 READ B$: PRINT B$,
7 RESTORE Bleistift
8 READ C$: PRINT C$
```

```
9 RESTORE
10 READ D$,E$,F$: PRINT D$,E$,F$

Katze     Hund      Radiergummi
Katze     Maus      Hund
```

RESUME

Typ: Befehl

Syntax: RESUME [{NEXT|<Marke>}]

Erklärung: RESUME beendet eine mit ON ERROR definierte Fehlerroutine. Der Befehl, der den Fehler ausgelöst hat, wird erneut ausgeführt. Bei RESUME NEXT wird die Zeile, in der der verursachende Befehl stand, übersprungen und in der darauffolgenden fortgefahren. Soll das Programm an anderer Stelle fortgesetzt werden, so ist auch die Angabe einer Sprungmarke möglich. RESUME mit Sprungmarke verläßt keine Strukturen oder Unterprogramme, die Sprungmarke muß also immer in derselben Hierachieebene liegen in der auch der Fehler erfolgte.

Beispiel:

```
 0 ON ERROR GOTO Fehler
 1 PRINT 1/0
 2 PRINT LN(-1)
 3 END
 4-Fehler
 5 MOUSEON
 6 FORM_ALERT (1,"[2]["+ ERR$ +"|in Zeile"+ STR$( ERL
                )+"][Nochmal|Weiter|Abbruch]",F_But)
 7 MOUSEOFF
 8 IF F_But=1 THEN RESUME
 9 IF F_But=2 THEN RESUME NEXT
10 IF F_But=3 THEN RESUME Fatal
11-Fatal
12 PRINT "Fataler Fehler erzwang Programmabbruch:"
13 PRINT ERR$ ;" in Zeile"; ERL
14 END
```

RETURN

Typ: Befehl

Syntax: RETURN [<Ausdruck>]

Erklärung: RETURN verläßt eine Prozedur oder ein Unterprogramm. RETURN mit Angabe eines Ausdrucks verläßt eine mehrzeilige Funktion und gibt den Ausdruck als Funktionswert zurück. Der Programmablauf wird an der Stelle fortgesetzt, von der aus der Aufruf erfolgt ist. Wird ein RETURN ausgeführt ohne daß ein Aufruf erfolgte, so wird ein "RETURN without GOSUB" ausgelöst.

Prozeduren können (und sollten) auch mit END_PROC abgeschlossen werden, wobei pro Prozedur allerdings nur ein END_PROC verwendet werden sollte, damit Anfang und Ende einer Prozedur klar definiert sind.

Ebenso steht für Funktionen der Befehl END_FN zu Verfügung. Dieser steht nur als Ende der Funktionsdefinition und gibt selbst keine Werte zurück. Trotzdem sollte er stets verwendet werden, um das Ende einer Funktion (speziell mit mehreren Ausgängen) klar zu kennzeichnen.

RETURN darf nicht innerhalb einer Struktur wie REPEAT-UNTIL, FOR-NEXT, WHILE-WEND oder SELECT-CASE aufgerufen werden. Es muß vielmehr immer zuerst die Struktur per EXIT verlassen und dann mit RETURN zurückgesprungen werden.

Beispiel:

```
0 PRINT FN Fib(10)
1 END
2 DEF FN Fib(N)
3   IF N>2 THEN
4     RETURN FN Fib(N-1)+FN Fib(N-2)
5   ELSE
6     RETURN 1
7   ENDIF
8 END_FN
 55
```

RIGHT$

Typ: Funktion

Syntax: RIGHT$(<Stringausdruck>,<num.Ausdruck>)

Erklärung: Ergibt einen Teilstring des Stringausdrucks mit der durch den numerischen Ausdruck gegebenen Länge. Der Teilstring endet am letzten Zeichen des Stringausdrucks.

Ist der numerische Ausdruck größer als die Länge des String-

Ausdrucks, so wird der gesamte String zurückgegeben.

Siehe auch LEFT$, MID$.

Beispiel:
```
Ø A$="OMIKRON.Software"
1 PRINT RIGHT$(A$,8)

Software
```

RMDIR

Typ: Befehl

Syntax: RMDIR <Stringausdruck>
RMDIR <Dateiname>

Erklärung: Entfernt einen Ordner, vorausgesetzt darin sind weder Dateien noch Unterordner enthalten.

RND

Typ: Funktion

Syntax: RND(<num.Ausdruck>)

Erklärung: Ergibt eine Zufallszahl in Abhängigkeit vom numerischen Ausdruck:

```
  Ø : Letzte Zufallszahl wiederholen
  1 : Wert von einschließlich Ø bis ausschließlich 1
sonst: ganzzahliger Wert zwischen Ø und dem numerischen
       Ausdruck, wobei der numerische Ausdruck nicht
       erreicht wird.
```

Beispiel:
```
PRINT RND(10)

?? (Zufallszahl)
```

RSET

Typ: Befehl

Syntax: RSET <String-Variable>=<Stringausdruck>

Erklärung: Der Stringausdruck wird rechtsbündig in String-Variable eingesetzt ohne daß deren Länge verändert wird. Hierzu wird der Stringausdruck vorne abgeschnitten oder durch Leerzeichen ergänzt.

Siehe auch LSET.

Beispiel:
```
Ø A$= SPACE$(20)
```

```
1 RSET A$="OMIKRON"
2 PRINT "*";A$;"*"

*              OMIKRON*
```

RUN

Typ: Befehl

Syntax: RUN [<Stringausdruck>] [,<Marke>]

RUN [<Dateiname>] [,<Marke>]

Erklärung: RUN startet das aktuelle Basic-Programm. Alle Variablen werden gelöscht und alle Strukturen des Programms geprüft. Das Programm kann nur gestartet werden, wenn die Strukturprüfung erfolgreich beendet wurde, ansonsten erscheint eine Fehlermeldung.

Mit der Angabe einer Marke kann man bestimmen, ab welcher Position im Programmtext das Programm gestartet wird.

Wird ein Programmname hinzugefügt, so wird zunächst das neue Programm geladen und dann gestartet. Das aktuelle Programm wird zuvor ohne Sicherheitsabfrage gelöscht.

Beispiel:

```
RUN Test ' startet ab der Marke "Test"
RUN 20 ' startet ab Zeile 20
RUN "DEMO" ' lädt und startet das Programm "DEMO.BAS"
```

SAVE

Typ: Befehl

Syntax: SAVE [<Stringausdruck>][,A]

SAVE [<Dateiname>][,A]

Erklärung: Das aktuelle Basic-Programm wird unter Dateiname gesichert. Ist kein Dateiname angegeben, so wird der alte Name benutzt. Das Programm kann auch als ASCII-Datei gespeichert werden, dann muß ,A angegeben werden. Siehe auch LOAD.

Wenn im Vollbildeditor die Option für automatisches Backup angewählt ist, erhält eine auf dem gleichen Pfad befindliche Datei mit gleichem Namen (altes Programm) die Extension .BAK. Ein altes .BAK wird dann gelöscht.

SCREEN

Typ: Befehl

Syntax: SCREEN <num.Ausdruck>

SCREEN <Bildschirmnummer>

Erklärung: Der SCREENbefehl reserviert Speicher für einen neuen Bildschirm. Auf diese Weise können mehrere Bildschirme von einem Programm abwechselnd benutzt werden. Mögliche Bildschirmnummern sind 1 - 2. Bildschirm 0 ist der Bildspeicher, der vom Programm und vom Direkt-Modus benutzt wird. Normalerweise wird nur der Textaufbau des Bildschirms 0 vermerkt, so daß ausgegebene Graphiken beim Wechsel in den Full-Screen-Editor verloren sind. Wenn Sie mit SCREEN 0 nun zusätzlich einen graphischen Bildspeicher einrichten, so werden auch alle Graphikausgaben beim Umschalten in den Direkt-Modus restauriert.

SCREEN 1 bzw. 2 reserviert Speicher für einen weiteren Bildschirm, den z.B. nur das Programm benutzt. Alle Ausgaben und Testausdrucke, die im Direkt-Modus getätigt wurden bleiben dadurch erhalten und bringen auch den Bildaufbau des Programms nicht durcheinander (der Direkt-Modus benutzt immer Bildschirm 0).

Beispiel:

```
0 LIBRARY Gem,"\GEM.LIB"
1 SCREEN 1
2 Appl_Init
3 STOP
4 Appl_Exit
5 END
```

Das GEM-Programm mit eigenem Bildschirmspeicher bricht in Zeile 2 ab und kehrt in den Direkt-Modus zurück. Im Direkt-Modus können nun zu Testzwecken Eingaben getätigt werden, ohne daß dies den Bildaufbau des Programms beeinflußt (z.B. "LIST"[Return]).

Durch gleichzeitiges Drücken beider Shift-Tasten kann man vorübergehend einen Blick auf das Schirmbild des Programms werfen. Mit CONT wird wieder in Bildschirm 1 umgeschaltet und das Programm fortgesetzt.

Gegenüber früheren Basic-Versionen sind weitere Parameter als die Bildschirmnummer beim Screenbefehl nicht zugelassen. Speicher für die zusätzlichen Bildschirme wird selbständig beim ersten Verwenden des SCREEN-Befehls reserviert (Speicherverbrauch: 32000 Bytes bei ST-Auflösungen, 153600 Bytes bei TT-Auflösungen). Bei Graphikkarten und Großbildschirmen errechnet sich der Speicherverbrauch entsprechend aus "Breite x Höhe x Anzahl Farbebenen".

Der Speicher wird vom GEMDOS-Speicher entnommen. Es muß also mit CLEAR für gesorgt werden, daß ausreichend Speicher frei ist, sonst wird mit "OUT OF MEMORY" abgebrochen (siehe CLEAR).

Beim nächsten CLEAR wird der Speicher automatisch wieder freigegeben.

Der SCREEN-Befehl schaltet nicht die logische oder gar physikalische Bildschirmadresse um, sondern kopiert per BITBLT hin- und her. Auf diese Weise ist die volle Kompatibilität gewährleistet.

SEC

Typ: Funktion

Syntax: SEC(<num.Ausdruck>)

Erklärung: Berechnet den Secans des numerischen Ausdrucks. Das Ergebnis ist abhängig vom eingestellten Winkelmodus (siehe DEG, RAD).

Beispiel:
```
RAD : PRINT SEC( .5)

 1.1394939
```

SECH

Typ: Funktion

Syntax: SECH(<num.Ausdruck>)

Erklärung: Berechnet den Secans Hyperbolicus des numerischen Ausdrucks.

Beispiel:
```
PRINT SECH(1)

 .64805427
```

SEEK

Typ: Befehl

Syntax: SEEK <Dateinummer>, <Position>[,<Modus>]

Erklärung: Mit dem SEEK-Befehl setzen Sie den internen Dateizeiger für Userdateien. Der nächste PUT- oder GET-Befehl schreibt bzw. liest dann ab dieser Position in der Datei. <Modus> hat folgende Bedeutung:

Modus	*Bedeutung*
keiner	Keine Modusangabe entspricht Modus 0
0	Die angegebene Position bezieht sich auf den Dateianfang.

1	Die Position ist relativ zur aktuellen Position gemeint. Auch negative Werte sind hier möglich (z.B. setzt -5 den Dateizeiger 5 Positionen zurück)
2	Die Position bezieht sich auf das Dateiende. Nur negative Werte oder 0 sind hier erlaubt.

SEGPTR

Typ: Funktion

Syntax: SEGPTR

Erklärung: Die Funktion SEGPTR liefert einen Zeiger auf eine interne Tabelle, die wiederum Zeiger auf verschiedene - vom Basic-Interpreter verwaltete - Speicherbereiche enthält. Die Benutzung dieser Zeiger sollte dem Spezialisten vorbehalten bleiben.

Offset	*Speicherbereich*
-12	BASEPAGE des Interpreters
0	Zeilen-Nummern-Tabelle
4	Start des BASIC-Programmes
8	Variablenpointer-Tabelle
12	Variablen-Tabelle
16	Reserviert
20	Arrays
24	Dateibuffer
28	Strings
32	Frei
36	Programm-Zeiger zeigt auf den gerade ausgeführten Befehl
40	Garbage-Top
44	Garbage-Bottom
48	Garbage-High
52	Stack-Maximalwert (unterhalb davon noch 1K für das Betriebssystem)
56	Stack-Bottom
60	Höchste Speicheradresse des BASIC (einstellbar mit CLEAR)

SELECT ... CASE ... END_SELECT

Typ: Befehl

Syntax: SELECT <Ausdruck> <Fall>[<Fall>...]{OTHERWISE|DEFAULT}... END_SELECT

mit <Fall>: CASE...[TO...] [{CONTINUE|EXIT}...[{CONTINUE|EXIT}..]]...

Erklärung: Mit ***SELECT*** wird die SELECT-CASE Anweisung eingeleitet. Der hinter SELECT stehende Ausdruck wird ausgewertet und auf eine Integer-Zahl abgebildet. Bei Stringausdrücken sind nur die ersten vier Zeichen von Belang. Abhängig von diesem einmal ermittelten Wert wird in die folgenden CASE-Teile verzweigt. Der Ausdruck wird nur einmal zu Beginn bewertet, d.h. wenn er später noch einmal modfiziert wird, hat dies keine Auswirkungen mehr auf die SELECT-CASE Verzweigungen.

CASE leitet innerhalb einer SELECT-CASE Anweisung eine Teilzweig ein. Dieser Programmzweig wird dann ausgeführt, wenn eine der genannten Möglichkeiten auf den bei SELECT genannten Ausdruck zutrifft. Mehrere Möglichkeiten werden einfach mit Kommata abgetrennt aufgezählt. Falls ein aufeinanderfolgender Bereich an Werten zutreffen soll, wird dieser mit TO verbunden. (z.Bsp.: CASE "A" TO "Z" ' prüft auf A,B,C,...,Y,Z)

CONTINUE innerhalb einer SELECT-CASE-Anweisung übergeht jeweils die nächste CASE-Bedingung und führt das Programm hinter dem CASE weiter aus.

Beispiel für CONTINUE:

```
Ø A$="Q"
1 SELECT A$
2   CASE "Q"
3     PRINT "Das Zeichen war ein Q"
4     CONTINUE
5   CASE "Ø" TO "9"
6     PRINT "Das Zeichen war eine Ziffer"
7 END_SELECT

Das Zeichen war ein Q
Das Zeichen war eine Ziffer
```

Innerhalb einer SELECT-CASE Anweisung wird mit ***DEFAULT*** oder ***OTHERWISE*** der Programmzweig eingeleitet, der ausgeführt werden soll, wenn keine der anderen Alternativen zutraf. Der DEFAULT- Zweig muß immer die letzte Möglichkeit innerhalb der SELECT-CASE Anweisung sein. Ein weiteres CASE darf nicht folgen.

END_SELECT schließlich beendet eine SELECT-CASE Anweisung. Wenn alle möglichen Alternativen aufgezählt sind, wird die SELECT-Anweisung mit END_SELECT abgeschlossen.

Beispiel:

```
0 INPUT "Wert 1-3 : ";Wert
1 SELECT Wert
2   CASE 1
3     PRINT "Der Wert war eins"
4   CASE 2
5     PRINT "Der Wert war zwei"
6   CASE 3
7     PRINT "Der Wert war drei"
8   DEFAULT
9   PRINT "Es sind nur Werte zwischen 1 und 3 zugelassen"
10 END_SELECT
11 END
```

SGN

Typ: Funktion

Syntax: SGN(<num.Ausdruck>)

SGN(X)

Erklärung: Die Signum-Funktion liefert:

-1 für X < 0

0 für X = 0

1 für X > 0

Beispiel:

```
PRINT SGN(7), SGN(-6), SGN(0)

 1        -1         0
```

SHL

Typ: Operator

Syntax: <num.Ausdruck> SHL <num.Ausdruck>

<num.Ausdruck> SHL <Anzahl>

Erklärung: Verschiebt bitweise nach links. Die höchstwertigsten Bits gehen verloren. Im niederwertigsten Bit wird eine Null nach geschoben. Anzahl muß zwischen 0 und 63 liegen sonst wird ein "Illegal function call" ausgelöst.

SHL vermittelt eine vorzeichenlose Integer-Multiplikation mit Zweierpotenzen ("X SHL Y" entspricht "X*(2^Y)").

Beispiel:

```
PRINT 4 SHL 2,1 SHL 10

 16          1024
```

SHR

Typ: Operator

Syntax: <num.Ausdruck> SHR <num.Ausdruck>

<num.Ausdruck> SHR <Anzahl>

Erklärung: Verschiebt bitweise nach rechts. Die niederwertigsten Bits gehen verloren. Im höchstwertigen Bit wird eine Null nach geschoben. Anzahl muß zwischen 0 und 63 liegen sonst wird ein "Illegal function call" ausgelöst.

SHR vermittelt eine vorzeichenlose Integer-Division durch Zweierpotenzen ("X SHR Y" entspricht "X\(2^Y)").

Beispiel:

```
PRINT 81 SHR 2,64 SHR 1

 20          32
```

SIN

Typ: Funktion

Syntax: SIN(<num.Ausdruck>)

Erklärung: Berechnet den Sinus des numerischen Ausdrucks. Das Ergebnis ist abhängig vom eingestellten Winkelmodus (siehe DEG, RAD).

SINH

Typ: Funktion

Syntax: SINH(<num.Ausdruck>)

Erklärung: Berechnet den Sinus Hyperbolicus des numerischen Ausdrucks.

Beispiel:

```
PRINT SINH(10)

 11013.233
```

SORT

Typ: Befehl

Syntax: SORT [ASC] <Feldbezeichner>(<num.Ausdruck>) [TO <Feldbezeichner>([<num.Ausdruck>])]

SORT [ASC] <Feldbezeichner>(<Minimum>) [TO <Feldbezeichner>(<Anzahl>)]

Erklärung: SORT sortiert ein eindimensionales Feld. Die Feldelemente werden in aufsteigender Reihenfolge angeordnet. Bei Stringfeldern wird streng nach DIN 5007 sortiert, d.h. Groß- und Kleinschreibung bleibt unberücksichtigt. Alle Umlaute werden wie die ausgeschriebenen Buchstabenkombinationen behandelt (Ä = AE, Ö = OE ...). Wenn nur nach ASCII-Werten sortiert werden soll, ist der Zusatz ASC anzugeben.

Folgt hinter dem SORT-Befehl ein TO mit einem weiteren Feldbezeichner, so wird dieses Feld mit umsortiert. Auf diese Weise können Indexfelder einfach verwaltet werden.

Anzahl gibt die Zahl der Elemente an, die sortiert werden sollen. Wird 0 angegeben oder der Anzahlparameter ganz weggelassen, so wird das gesamte Feld sortiert.

Beispiel:

```
0  DIM Werte(10)
1  FOR I=0 TO 10
2    Werte(I)= RND(10)
3  NEXT I
4  '
5  SORT Werte(0) ' 0 = Alles
6  '
7  FOR I=0 TO 10
8    PRINT Werte(I)
9  NEXT I
```

Beispiel:

```
 0 DIM Texte$(11)
 1 '
 2 REPEAT
 3   INPUT "beliebiger Text [Return]->ENDE : ";Texte$(N)
 4   IF LEN(Texte$(N))=0 THEN EXIT
 5   N+=1
 6 UNTIL N>10
 7 '
 8 SORT Texte$(N)
 9 '
10 FOR I=0 TO N
```

```
11   PRINT Texte$(I)
12 NEXT I
```

SPACE$

Typ: Funktion

Syntax: SPACE$(<num.Ausdruck>)

Erklärung: Ergibt einen mit Leerzeichen (CHR$(32)) gefüllten String mit der durch den numerischen Ausdruck gegebenen Länge. Die Funktion ist identisch zu SPC.

Siehe auch STRING$.

Beispiel:
```
PRINT "*";SPACE$(14);"*"

*              *
```

SPC

Typ: Funktion

Syntax: SPC(<num.Ausdruck>)

Erklärung: Ergibt einen mit Leerzeichen (CHR$(32)) gefüllten String mit der durch den numerischen Ausdruck gegebenen Länge. Die Funktion ist identisch zu SPACE$.

Siehe auch STRING$.

Beispiel:
```
PRINT "*";SPC(14);"*"

*              *
```

SQR

Typ: Funktion

Syntax: SQR(<num.Ausdruck>)

Erklärung: Berechnet die Quadratwurzel des numerischen Ausdrucks. Der numerische Ausdruck muß ein positiver Wert sein.

Beispiel:
```
PRINT SQR(2.), SQR(2#)

 1.4142136        1.4142135623730951
```

STEP

Erklärung: Schrittweite bei FOR-Schleifen. Siehe FOR

STOP

Typ: Befehl

Syntax: STOP

Erklärung: STOP bricht die Programmausführung mit der Fehlermeldung "?Break in ..." ab. Der Interpreter schaltet automatisch in den Direktmodus. Es ist nun möglich beliebige Befehle direkt auszuführen, um so z.B. Variableninhalte (DUMP), bestimmte Feldelemente oder Speicherbereiche zum Zweck der Fehlersuche anzuschauen. Auch ein Wechseln in den Vollbildeditor ist möglich. Das Programm kann jederzeit mit CONT wieder fortgesetzt werden, solange das Programm nicht geändert wurde.

Siehe auch CONT.

Beispiel:

```
0 REPEAT
1   IF RND(20)>18 THEN STOP
2   Count+=1
3 UNTIL 0 ' endlos

? Break in 1
```

STR$

Typ: Funktion

Syntax: STR$(<num.Ausdruck>)

Erklärung: STR$ wandelt den numerischen Ausdruck in einen String um. STR$ ist also das Gegenstück zu VAL. Bei der Umwandlung werden alle mit USING getroffenen Einstellungen beachtet. Der erzeugte String sieht also genauso aus, wie das, was PRINT USING <num. Ausdruck> ausgeben würde.

STR$ wandelt immer ins Dezimal-Format, gegebenenfalls in Exponentialdarstellung. Zur Wandlung in andere Zahlensysteme verwenden Sie HEX$, OCT$ oder BIN$.

Siehe auch VAL, USING, HEX$, OCT$, BIN$.

Beispiel:

```
0 Pi$=STR$(PI)
1 USING " DM ###.##"
2 Betrag$=STR$(12.121)
```

```
3 USING ' abschalten
4 PRINT Pi$,Betrag$

 3.1415926535897932      DM  12.12
```

STRING$

Typ: Funktion

Syntax: STRING$(<num.Ausdruck>,{<num.Ausdruck>|<Stringausdruck>})

STRING$(<Anzahl>,{ASCII-Code|<Stringausdruck>}

Erklärung: Erzeugt einen String der Länge Anzahl, in dem das durch String-Ausdruck oder seinem entsprechenden ASCII-Code (siehe ASCII-Tabelle) gegebene Zeichen entsprechend oft vorkommt. Es gilt nur das erste in Stringausdruck vorkommende Zeichen. Siehe auch SPACE$, SPC, *.

Beispiel:

```
PRINT STRING$(10,"*"),STRING$(10,240)

**********
```

SWAP

Typ: Befehl

Syntax: SWAP <Variable>,<Variable>

Erklärung: Vertauscht die Inhalte der beiden gegebenen Variablen. Die beiden Variablen müssen identischen Variablentyps sein. Ein Tausch einer Feldvariable mit einer nicht-Feldvariable ist nur beim Typ Integer-Langwort erlaubt.

Beispiel:

```
0 A=10:B=20
1 SWAP A,B
2 PRINT A,B

 20        10
```

SYSTEM

Typ: Befehl

Syntax: SYSTEM

Erklärung: Verläßt OMIKRON.Basic und kehrt zum aufrufenden Programm zurück. Vor der tatsächlichen Ausführung wird der Benutzer aufgefordert, den Befehl zu bestätigen. Enthält diese Sicherheitsabfrage den Zusatz "Changes will be lost" so wurde das aktuelle Programm noch nicht gesichert. Wenn Sie trotzdem fortfahren ist die letzte Änderung bzw. das aktuelle Programm verloren.

Beispiel:
```
SYSTEM
Quit BASIC - Changes will be lost - sure (Y/N)?
```

TAB

Typ: PRINT-Funktion

Syntax: TAB(<num.Ausdruck>)

Erklärung: Diese Funktion tabuliert Ausgaben von PRINT, LPRINT oder PRINT #. Der numerische Ausdruck gibt die neue Ausgabespalte an auf die der Schreibzeiger vorrückt. Es werden soviele Leerzeichen ausgegeben, bis die gewünschte Spalte erreicht ist.

Siehe auch @, SPC, PRINT.

Beispiel:
```
Ø PRINT TAB(1Ø);"1Ø. Spalte";TAB(25);"25. Spalte"
1 PRINT TAB(1Ø);1.256;TAB(25);3.6589

          1Ø. Spalte      25. Spalte
           1.256           3.6589
```

TAN

Typ: Funktion

Syntax: TAN(<num.Ausdruck>)

Erklärung: Berechnet den Tangens des numerischen Ausdrucks. Das Ergebnis ist abhängig vom eingestellten Winkelmodus (siehe DEG, RAD).

Beispiel:
```
RAD: PRINT TAN(PI / 8)

 .41421356237309505
```

TANH

Typ: Funktion

Syntax: TANH(<num.Ausdruck>)

Erklärung: Berechnet den Tangens Hyperbolicus des numerischen Ausdrucks.

Beispiel:
```
PRINT TANH(.5)

 .46211716
```

TEXT

Typ: Befehl

Syntax: TEXT <num.Ausdruck>,<num.Ausdruck>,<Stringausdruck>[,<num.Ausdruck>,<num.Ausdruck>,<num.Ausdruck>]

TEXT <X>,<Y>,<Ausgabetext>[,<Breite>,<Wortflag>,<Buchstabenflag>]

Erklärung: Dieser Befehl gibt einen formatierten Grafik-Text. Die Koordinaten X und Y erlauben pixelgenaues Positionieren des Textes. Die Koordinaten beziehen sich immer auf die linke untere Ecke des Textes.

Wenn Textbreite und Formatflags mit angegeben sind, wird je nach Einstellung zusätzlich Formatiert. Ist mindestens eines der beiden Flags auf eins gesetzt so werden Wort- oder Buchstabenzwischenräume gedehnt, so daß ein ausgeglichener Blocksatz entsteht.

Textfarbe, Stil, Größe und Ausrichtung können über TEXT COLOR, TEXT STYLE, TEXT HEIGHT bzw. TEXT ROTATION bestimmt werden.

Ist mittels CLIP ein Bildfenster definiert, wird außerhalb dieses Bereiches nicht gezeichnet.

Beispiel:
```
0 TEXT 10,23,"Dieser lange Satz soll genau in den Kasten
                                              passen",420,1,0
1 BOX 9,9,422,18
2 PRINT : PRINT
```

TEXT COLOR

Typ: Befehl

Syntax: TEXT COLOR = <num.Ausdruck>

TEXT COLOR = <Farbnummer>

Erklärung: Definiert die Schriftfarbe für Grafik-Text.

TEXT HEIGHT

Typ: Befehl

Syntax: TEXT HEIGHT = <num.Ausdruck>

TEXT HEIGHT = <Höhe in Pixeln>

Erklärung: Setzt die Text-Höhe des Grafik-Textes. Die Höhenangabe in Punkten wird vom VDI anhand der Höhenangabe im Dateikopf der Systemzeichensätze umgerechnet. Eine Punktgrösse von 4 ergibt die 6x6 Schrift, 6 die 8x8 Schrift,13 die 8x16 Schrift und 26 die 16x32 Schrift (standardmäßig nur auf ATARI TT vorhanden). Ist eine passende Schriftgrösse nicht direkt als fertiges Pixelmuster verfügbar, versucht das VDI durch Vergößern oder Verkleinern einer benachbarten Schriftgröße die Lücke zu schließen. Das Ergebnis zeigt das Beispielprogramm: es kann durchaus vorkommen, daß eine 12 Punkt Schrift größer ist als eine 13 Punkt Schrift.

Negative Werte stellen die Texthöhe mit Hilfe des GEM VDI- Befehls vst_point()ein. Dabei werden nur ganzzahlige Vergrößerungen der Fonts verwendet.

Beispiel:

```
0 FOR Text_H=2 TO 17
1   TEXT HEIGHT =Text_H
2   TEXT 10,Y+5, STR$(Text_H)+" Punkt Textprobe"
3   Y+=Text_H*2
4 NEXT Text_H
```

TEXT ROTATION

Typ: Befehl

Syntax: TEXT ROTATION = <num.Ausdruck>

TEXT ROTATION = <Winkel>

Erklärung: Legt den Neigungswinkel für Grafiktext fest. Abhängig vom Workstationtreiber ist die Einstellung des Winkels nur eingeschränkt möglich. Der Standard-Bildschrim-Treiber für ATARI ST/TT kann z.B. nur 90 Grad-Schritte. Die Winkelangabe erfolgt in 1/10 Grad (d.h. 900 entspricht 90 Grad).

Beispiel:

```
TEXT ROTATION = 2700
TEXT 10,10,"Der Text geht nach unten"
```

TEXT STYLE

Typ: Befehl

Syntax: TEXT STYLE=<num.Ausdruck>

TEXT STYLE=<Stil-Attribut>

Erklärung: Definiert die Stil-Attribute des mit TEXT auszugebenden Grafiktextes. Dabei steht jedes Bit in Stil-Attribut für einen Texteffekt. Ist es gesetzt, so wird der Effekt verwendet. Beliebige Kombinationen sind zulässig.

Bit	Wertigkeit	Effekt
0	1	fett
1	2	hell
2	4	kursiv
3	8	unterstrichen
4	16	outline (Umrißlinie)

Die TEXT STYLE Einstellung betrifft nur die Ausgaben mit dem Befehl TEXT. Auf den Befehl PRINT haben sie keinen Einfluß.

Beispiel:

```
0 TEXT STYLE = 1
1 TEXT 10,10,"Dieser TEXT erscheint fett"
2 TEXT STYLE = 4 + 16
3 TEXT 10,20,"Dieser TEXT hingegen kursiv & outlined"
```

THEN

Erklärung: Siehe IF

TIME$

Typ: Befehl

Syntax: TIME$=<Stringausdruck>

Erklärung: Weist der Systemuhr die Zeit Stringausdruck zu.

Stringausdruck ist unabhängig vom Ländermodus (siehe MODE) anzugeben (0 Uhr bis 23:59:59):

"HH:MM:SS"

(H=Stunde, M=Minute, S=Sekunde)

Die mit TIMER ermittelbare Rechner-Laufzeit wird nicht beeinflußt.

TIME$

Typ: Funktion

Syntax: TIME$

Erklärung: Gibt die Systemuhr in folgender Form als String:

"HH:MM:SS"

(H=Stunde, M=Minute, S=Sekunde)

Das Ergebnis ist nicht abhängig vom über MODE eingestellbaren Ländermodus.

TIMER

Typ: Funktion

Syntax: TIMER

Erklärung: Ergibt die Laufzeit des Rechners seit dem Einschalten in 1/200 stel Sekunden.

Beispiel:

```
0 T=TIMER
1 FOR I=0 TO 100000
2 NEXT I
3 PRINT (TIMER-T)/200

 3.675
```

TO

Erklärung: Siehe FOR, SELECT (,BOX, DRAW, NDC)

TROFF

Typ: Befehl

Syntax: TROFF

Erklärung: TROFF schaltet den TRACE-Modus des Interpreters ab.

WICHTIG: Innerhalb des Programms kann TRON und TROFF geschachtelt werden, d.h. pro Einschaltvorgang durch TRON ist auch ein Ausschaltvorgang durch TROFF notwendig.

Im Direkt-Modus wird durch TROFF der TRACE-Modus in jedem Fall ausgeschaltet.

Siehe auch TRON.

TRON

Typ: Befehl

Syntax: TRON

Erklärung: TRON schaltet den TRACE-Modus des Interpreters ein. Dadurch wird vor der Ausführung eines jeden Befehls die Zeilennummer und der Befehl am Bildschirm ausgegeben. Wenn Sie den Programmablauf über weite Strecken kontrollieren möchten, achten Sie darauf, daß der automatische Zeilenumbruch eingeschaltet ist. Ansonsten staut sich quasi die Ausgabe am rechten Bildschirmrand. Den automatischen Umbruch aktivieren Sie mit PRINT CHR$(27);"v";.

Wurde mit ON TRON GOSUB ein spezielles TRACE-Unterprogramm definiert so wird bei jedem Befehl dieses Unterprogramm durchlaufen. TROFF schaltet den TRACE-Modus wieder ab und setzt die normale Programmausführung fort.

WICHTIG: Innerhalb des Programms kann TROFF und TRON geschachtelt werden, d.h. pro Schaltvorgang durch TROFF ist auch ein Schaltvorgang durch TRON notwendig.

Siehe auch VT-52-Liste, ON TRON GOSUB, TROFF.

Beispiel:

```
0 TRON:PRINT CHR$(27);"v";
1 REPEAT
2 UNTIL LEN(INKEY$)
3 END
```

TUNE

Typ: Befehl

Syntax: TUNE <num.Ausdruck>,<num.Ausdruk>[[,<num.Ausdruck>,<num.Ausdruk>]]

TUNE <Tonkanal>,<Frequenz>[[,<Tonkanal>,<Frequenz>]]

Erklärung: Den angegebenen Tonkanälen wird die Frequenz zugewiesen. Tonkanal muß ein Wert von 1 bis 3 sein, Frequenz muß ein Wert von 0 bis 4095 sein. 0 schaltet den Ton ab, 1 ist der höchste und 4095 der tiefste Ton.

Um den Ton zu hören, muß den entsprechenden Kanälen mittels VOLUME eine Lautstärke zugewiesen werden.

Siehe auch NOISE.

Beispiel:

```
0 Conterm=$484
1 Conterm_Rett= PEEK(Conterm)
2 POKE Conterm,0
3 T=0:Dt=1
4 VOLUME 1,15
5 REPEAT
6   TUNE 1,T
7   T+=Dt
```

```
8   IF T=4095 OR T=0 THEN Dt*=-1
9  UNTIL LEN( INKEY$ )
10 VOLUME 1,0
11 POKE Conterm,Conterm_Rett
```

UNTIL

Typ: Befehl

Syntax: UNTIL <log. Ausdruck>

UNTIL <Abbruch-Bedingung>

Erklärung: Beendet die REPEAT-UNTIL-Schleife. Ist die Abbruch-Bedingung nicht erfüllt, so wird die Schleife noch einmal ausgeführt.

Siehe auch REPEAT-UNTIL.

UNLIST

Typ: reserviert

UPPER$

Typ: Funktion

Syntax: UPPER$(<Stringausdruck>)

Erklärung: Wandelt alle im Stringausdruck enthaltenen Buchstaben in große Buchstaben um. Auch die Umlaute ä, ö, ü werden gewandelt.

Gegenstück zu dieser Funktion ist LOWER$.

USING

Typ: Befehl

Syntax: USING [<Stringausdruck>]

Erklärung: Stellt das Ausgabeformat für die Befehle PRINT, LPRINT, PRINT # und die Funktion STR$ ein. USING allein ohne Stringausdruck schaltet wieder auf normale Ausgabe zurück. Welche Zeichen der Stringausdruck enthalten darf und wie diese wirken sehen Sie bitte unter PRINT USING nach.

[L]PRINT[#] USING wirkt nur vorübergehend, d.h. es zerstört ein globales USING nicht. PRINT USING;xxx schaltet das globale USING vorübergehend ab.

USR

Typ: Funktion

Syntax: USR(<num.Ausdruck>)

Erklärung: Die mit DEF USR definierte Maschinensprachefunktion wird aufgerufen. Der Parameter wird im Register D0 übergeben. Der im Register D0 erhaltene Wert wird als Funktionsergebnis an das Basic-Programm zurückgegeben.

VAL

Typ: Funktion

Syntax: VAL(<Stringausdruck>)

Erklärung: Der Stringausdruck wird nach Möglichkeit in ein numerisches Ergebnis umgewandelt. Dabei wird solange von links nach rechts gelesen, bis kein gültiges Zahlzeichen mehr folgt. Führende Leerzeichen werden überlesen.

Gültige Zahlzeichen sind: $ & % 0 1 2 3 4 5 6 7 8 9 + - . D E

VAL kann durch hinzufügen der Prefixe "$" oder "%" auch zum Wandeln zwischen verschiedenen Zahlensystemen benutzt werden (z.B. HEX -> Dezimal).

Beispiel:

```
Ø Hex_In$="Øu+A+B+C+D+E+F"
1 INPUT "Geben Sie eine HEX-Zahl ein : ";Zahl$ USING Hex_In$,,2
2 PRINT
3 Zahl$= RIGHT$("ØØØ"+Zahl$,3)
4 PRINT Zahl$;"h ist Dezimal: "; VAL("$"+Zahl$)
```

VARPTR

Typ: Funktion

Syntax: VARPTR(<Variable>)

Erklärung: Die Funktion entspricht exakt dem Adress-Operator & und ermittelt einen Zeiger auf die Variable. Bei Feldvariablen haben die Indices keine Bedeutung und können deshalb entfallen.

Dieser Zeiger stellt jedoch nicht in jedem Fall direkt die Adresse der zugehörigen Variablen dar. Zudem werden Variablen vom Interpreter und Compiler unterschiedlich verwaltet. Deshalb sollte der Zugriff über diesen Zeiger immer über den Zeiger-Operator * erfolgen. Direktes Verwenden oder gar Manipulieren des Zeigers erfordert eine genaue

Kenntnis der Speicherverwaltung von Interpreter und Compiler (siehe Anhang).

Praktische Anwendung: Mit VARPTR(MOUSEX) erhält man die Adresse im Speicher, ab der Mauskoordinaten verwaltet werden. Dort können Sie auch geändert werden. VARPTR wird auch verwendet, um Speicheradressen von Strings, Variablen oder Feldern zu erhalten.

Beispiel:

```
 0 ' größtes Element suchen
 1 DIM Werte#(10)
 2 PRINT FN Get_Max#(VARPTR(Werte#()),5) ' nur erste 5
 3 END
 4 DEF FN Get_Max#(Pointer,N)
 5   LOCAL I,Maximum#=0#
 6   FOR I=0 TO N
 7     Maximum#=MAX(*Pointer#(I),Maximum#)
 8   NEXT I
 9   RETURN Maximum#
10 END_FN

0
```

VDI

Typ: Befehl

Syntax: VDI(<num.Ausdruck>, <Feldvariable>, <Feldvariable>, <Feldvariable>, <Feldvariable>, <Feldvariable>)

VDI(<Funktionsnummer>, <CTRL-Feld>, <INTIN-Feld>, <PTSIN-Feld>, <INTOUT-Feld>, <PTSOUT-Feld>)

Erklärung: Mit dem Befehl VDI rufen Sie GEM VDI (Graphics Environment Manager - Virtual Device Interface) auf. Die einzelnen Funktionen des GEM VDI schlagen Sie bitte im Anhang nach.

Für VDI- und AES-Befehle benutzen Sie einfacher die GEM- oder EasyGEM-Library. In der Funktionsnummer sind Flags für NDC enthalten. Allgemein gilt: VDI ist der GEMLIB vorbehalten.

VERSION

Typ: Funktion

Syntax: VERSION

Erklärung: Liefert die Versionsnummer des verwendeten Interpreters. 300 steht für

3.00, 351 für 3.51 etc. In compilierter Fassung liefert VERSION immer einen Wert >=999, da ihr Compiler (hoffentlich) immer Up-to-date ist.

VOLUME

Typ: Befehl

Syntax: VOLUME <num.Ausdruk>,{<num.Ausdruck>|<num.Ausdruck>,<num.Ausdruck>}

VOLUME <Tonkanal>,{<Lautstärke>|<Hüllkurve>,<Intervall>}

Erklärung: Dem angegebenen Tonkanal wird eine Lautstärke oder eine Hüllkurve und deren Intervall zugewiesen. Tonkanal muß ein Wert von 1 bis 3 sein, Lautstärke von 0 (Ton aus) bis 15 (maximale Lautstärke).

Wird statt der Laustärke Hüllkurve gegeben, so sind folgende Werte möglich:

8: an, fallend, an, fallend, ...

9: an, fallend, aus

10: an, fallend, steigend, fallend, steigend, ...

11: an, fallend, an

12: aus, steigend, aus, steigend, ...

13: aus, steigend, an

14: aus, steigend, fallend, steigend, fallend, ...

15: aus, steigend, aus

Intervall muß ein Wert zwischen 0 und 65535 sein. Je kleiner Intervall ist, desto schneller läuft die Hüllkurve ab.

Bei Verwendung einer Hüllkurve werden die Lautstärken aller drei Tonkanäle gleichzeitig gesteuert.

Bei der Verwendung von Sound-Befehlen sollte der Tastaturklick mittels der Betriebsystemvariable comterm abgeschaltet werden.

Beispiel:

```
0 Conterm=$484
1 Conterm_Rett= PEEK(Conterm)
2 POKE Conterm,0
3 V=0:Dv=1
4 TUNE 1,1000
5 REPEAT
6   WAIT .02
7   VOLUME 1,V
8   V+=Dv
9   IF V=15 OR V=0 THEN Dv*=-1
10 UNTIL LEN( INKEY$ )
```

```
11 VOLUME 1,0
12 POKE Conterm,Conterm_Rett
```

WAIT

Typ: Befehl

Syntax: WAIT <num.Ausdruck>

Erklärung: Wartet die im numerischen Ausdruck genannte Zeit in Sekunden. Der Fehler beträgt +/- 1/200 stel Sekunde.

Die Multitasking-Funktionen ON HELP GOSUB, ON KEY GOSUB, ON MOUSEBUT GOSUB und ON TIMER GOSUB werden jedoch währenddessen weiterbearbeitet.

Beispiel:

```
0 T=TIMER
1 REPEAT
2   PRINT "Diese Schleife läuft genau 6 Sekunden"
3   WAIT .5
4 UNTIL TIMER>T+200*6
```

WEND

Typ: Befehl

Syntax: WEND

Erklärung: Beendet die WHILE-WEND Schleife. Siehe WHILE-WEND.

WHILE ... WEND

Typ: Befehl

Syntax: WHILE <num.Ausdruck> : <Befehle> : WEND

WHILE <log. Ausdruck> : <abhängige Befehle> : WEND

Erklärung: Mit WHILE wird eine Programmschleife eingeleitet, die solange, die von ihr abhängigen Befehle wiederholt, bis der logische Ausdruck (die Bedingung) nicht mehr erfüllt (falsch, gleich 0) ist. Die Bedingung wird immer am Anfang der Schleife geprüft, d.h. die WHILE-WEND-Schleife ist eine abweisende Schleife. Wenn die Bedingung gleich zu Anfang nicht erfüllt ist wird keine der abhängigen Befehle ausgeführt und zum Ende der Schleife (hinter das WEND) verzweigt.

Beispiel:

```
0 OPEN "I",1,"C:\DESKTOP.INF"
1 WHILE NOT EOF(1)
```

```
 2   PRINT INPUT$(1,1);
 3 WEND
 4 CLOSE 1
 5 PRINT "Bitte geben sie mehrere Wörter durch
                    Leerzeichen getrennt ein: "
 6 INPUT Satz$
 7 P=INSTR(Satz$," ")
 8 WHILE P
 9   PRINT "Leerzeichen gefunden an Position:"; P
10   P=INSTR(P,Satz$," ")
11 WEND
```

WPEEK

Typ: Funktion

Syntax: WPEEK(<num.Ausdruck>)

Erklärung: Liest an der durch numerischen Ausdruck gegebenen Adresse ein Wort. Der numerische Ausdruck muß eine gerade Zahl sein, da

sonst ein Bus-Fehler auftritt. Der gelesene Wert wird immer als Zweierkomplement-Darstellung einer Integer-Wort-Zahl aufgefaßt und liegt im Bereich von -32768 bis +32767.

Siehe auch PEEK, LPEEK, POKE, WPOKE, LPOKE.

Beispiel:

```
0 Speicher=MEMORY(10)
1 WPOKE Speicher,$FFFF
2 PRINT WPEEK(Speicher)
```

```
 -1
```

WPOKE

Typ: Befehl

Syntax: WPOKE <num.Ausdruck>,<num. Ausruck>

WPOKE <Addresse>,<Wert>

Erklärung: Legt <Wert> an der gegebenen Adresse als ein Langwort ab. Der numerische Ausdruck muß auf 68000ern (ATARI ST) eine gerade Zahl sein, da

sonst ein Bus-Fehler auftritt. Auf 68030-Prozessorn (ATARI TT) kann die Adresse ungerade sein.

Der Wert muß zwischen 0 und 65535 liegen. Negative Werte größer-gleich -32768 sind auch zulässig und gelten als Zweier-Komplement (-1=65535, -2=65534, -3=65533, ...).

Siehe auch POKE, LPOKE, PEEK, WPEEK, LPEEK.

Beispiel:

```
0 Speicher=MEMORY(10)
1 WPOKE Speicher,$FFFF
2 WPOKE Speicher+2,-1
3 WPOKE Speicher+4,65500
```

WRITE

Typ: Befehl

Syntax: WRITE <Ausdruck>[[,<Ausdruck>]]

WRITE <Ausgabe>[[,<Ausgabe>]]

Erklärung: WRITE gibt wie PRINT beliebige Ausdrücke auf dem Bildschirm aus.

Stringausdrücke werden in Anführungszeichen gesetzt und alle Ausgaben durch Kommata getrennt.

Im Gegensatz zum PRINT-Befehl (s. dort) werden bei diesem Befehl alle Zeichen genauso ausgegeben wie Sie sie geschrieben haben. Anführungszeichen, Kommata, Strichpunkte usw. dienen also nicht als Trennzeichen, sondern werden direkt ausgegeben.

Beispiel:

```
0 Test$="OMIKRON.Software"
1 WRITE Test$,PI,1.23

"OMIKRON.Software", 3.1415926535897932, 1.23
```

WRITE

Typ: Befehl

Syntax: WRITE #<num.Ausdruck>,<Ausdruck>[[,<Ausdruck>]]

WRITE #<Dateinummer>,<Ausgabe>[[,<Ausgabe>]]

Erklärung: WRITE # gibt - ähnlich wie PRINT # - beliebige numerische oder String-Ausdrücke in eine Datei aus. Die Datei muß zuvor mit OPEN geöffnet worden sein. Im Vergleich zu PRINT # setzt WRITE # alle Stringausdrücke in Anführungszeichen und trennt alle Ausdrücke durch Kommata ab. Das hat besonders beim Schreiben von Dateien, die mit INPUT # wieder gelesen werden sollen, große Vorteile.

Siehe auch PRINT #, INPUT #

Beispiel:

```
0 Test$="OMIKRON.Software"
```

```
1 OPEN "C",1
2   WRITE #1,Test$,PI,1.23
3 CLOSE 1

"OMIKRON.Software", 3.1415926535897932, 1.23
```

WVBL

Typ: Befehl

Syntax: WVBL

Erklärung: Wartet auf einen Vertical-Blank-Interrupt.

Die Multitasking-Funktionen ON HELP GOSUB, ON KEY GOSUB, ON MOUSEBUT GOSUB und ON TIMER GOSUB werden jedoch währenddessen weiterbearbeitet.

W_CHAR

Typ: Funktion

Syntax: W_CHAR

Erlärung: Ergibt die Breite des Bildschirms in Zeichen. Der Wert entspricht den Informationen, die vom VDI bei "Inquire Character Cells" geliefert werden. Siehe auch H_CHAR.

W_PIXEL

Typ: Funktion

Syntax: W_PIXEL

Erklärung: Ergibt die Breite des Bildschirms in Bildpunkten (Pixeln). Der Wert entspricht den Informationen, die vom VDI bei V_Openvwk geliefert werden. Siehe auch H_CHAR und H_PIXEL.

XBIOS

Typ: Befehl

Syntax: XBIOS[(][<num.Variable>],<num.Ausdruck>
[[,[L]<num.Ausdruck>]][)]

XBIOS[()[<Rückgabe-Variable>],<Funktionsnummer>[[,
[L]<Parameter>]][)]

Erklärung: Die in Funktionsnummer genannte XBIOS-Funktion wird aufgerufen und die Parameter übergeben. Der Rückgabe-Wert der XBIOS-Funktion wird der Rückgabe-Variablen zugewiesen. Wenn vor den Parametern ein "L " gestellt ist, so wird der Parameter als LONG übergeben, ansonsten immer als WORD.

Siehe auch XBIOS-Funktionsliste.

Beispiel:

```
0 PRINT HEX$(FN Logbase)
1 END
2 DEF FN Logbase
3   LOCAL R=0
4   XBIOS R,3
5   RETURN R
6 END_FN
```

XOR

Typ: Operator

Syntax: <num.Ausdruck> XOR <num.Ausdruck>

Erklärung: Die beiden Ausdrücke werden bitweise "logisch exklusiv-oder" verknüpft.

Beispiel:

```
0 PRINT BIN$((%1010 XOR %1100)+%10000)

 10110
```

Farben und Muster

VDI-Farbe	Hardware-Farbe		
	16-Farb	4-Farb	Schwarz-Weiss
0	0	0	0
1	15	3	1
2	1	1	
3	2	2	
4	4		
5	6		
6	3		
7	5		
8	7		
9	8		
10	9		
11	10		
12	12		
13	14		
14	11		
15	13		

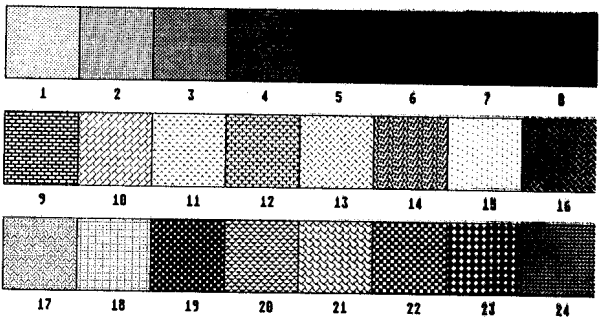
FILL INTERIOR STYLES

Hier eine Übersicht über die FILL STYLE-Parameter und die dadurch eingestellten Muster. (siehe auch FILL STYLE, PBOX, PCIRCLE etc.)

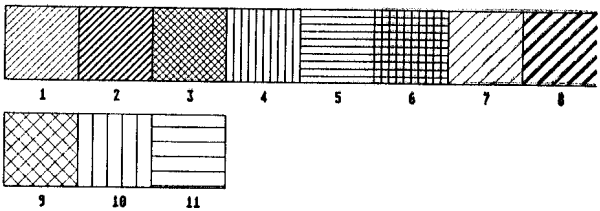
FILL STYLE = 0,

FILL STYLE = 1,

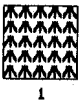
FILL STYLE = 2,



FILL STYLE = 3,



FILL STYLE = 4,



Dateitypen (Open)

Der beim OPEN-Befehl angegebene Dateityp gibt an, um was für eine Art von Datei es sich handeln soll. Als Dateitypen dürfen angegeben werden:

"I" "Input" Sequentielle (Disketten)datei; nur zu lesen

Beispiel: OPEN "I",1,"A:\DESKTOP.INF"

MIT INPUT#1,... können Sie Daten auslesen, EOF(1) ermittelt, ob das Dateiende erreicht ist.

"O" "Output" Sequentielle (Disketten)datei; nur zu schreiben

Beispiel: OPEN "O",1,"ADRESSEN.DAT"

Daten schreibt man mit PRINT#1 oder WRITE#1.

"A" "Append" wie "Output", fügt jedoch an eine bereits bestehende Datei an:

Beispiel: OPEN "A",1,"ADRESSEN.DAT"

"R" "Random" Random-Access-(Disketten)datei

Beispiel: OPEN "R",1,"STAMM.DAT",142: FIELD 1,...

Die Zahl 142 ist die "Recordlänge" der Datei, also die Länge eines Datensatzes.

Die FIELD-Anweisung legt den Datensatz fest, GET und PUT lesen und schreiben Daten, LOF ermittelt die Größe der Datei.

"F" "Files" Inhaltsverzeichnis lesen

Beispiel: OPEN "F",1,"A:.*",63*

Die Zahl 63 ist eine Bitmaske, die angibt, was für Dateiattribute bei der Suche zugelassen werden sollen. Die 63 errechnet sich aus:

1 für schreibgeschützte Dateien

+2 für verborgene Dateien

+4 für Systemdateien

+8 für Diskettenname

+16 für Ordner

+32 für Archivbit gesetzt (erst ab TOS 1.04!)

Mit GET 1,0 wird der nächste Eintrag geholt, mit EOF(1) können Sie ermitteln, ob der Eintrag gültig war. (Siehe auch Programm "FILES.BAS")

"P" "Printer" eröffnet eine sequentielle Datei auf den Drucker

Beispiel: OPEN "P",1

Man kann nun mit PRINT#1 auf den Drucker ausgeben.

"V" "V24" eröffnet eine sequentielle Datei zur RS232-Schnittstelle.

Beispiel: OPEN "V",1

PRINT#1 schreibt Daten über die RS232-Schnittstelle, INPUT#1 liest Daten von RS232.

"M" "MIDI" eröffnet eine sequentielle Datei zur MIDI-Schnittstelle.

Beispiel: OPEN "M",1

"K" "Keyboard" ist nicht mehr vorhanden. Siehe "OPEN".

Dateinamen

Ein Dateiname besteht aus drei Teilen:

- Der Laufwerkname, z.B. A:
- Der Pfadname, z.B. \AUTO\
- Der Dateiname, z.B. ADRESSEN.DAT

Der Laufwerkname ist ein Name von "A:" bis "P:". Wenn Sie Ihn weglassen, so bezieht sich die Datei automatisch auf das Standardlaufwerk, also das Laufwerk, von dem aus der Interpreter geladen wurde. (Ausnahme: von Laufwerk O: gestartet, setzt OMIKRON.BASIC das Standardlaufwerk auf A: oder auf C: - falls vorhanden - um.) Das Standardlaufwerk können Sie mit z.B. CHDIR "B:" umstellen.

Ist das erste Zeichen des Pfadnamens ein Rückwärts-Schrägstrich (Sie erreichen ihn mit [Alternate]-[Shift]-[\]), so wird vom obersten (Haupt-) Inhaltsverzeichnis ausgegangen, ansonsten vom Standardpfad des Laufwerks. Für jedes Laufwerk gibt es einen eigenen Standardpfadnamen.

Beispiel: `CHDIR "A:\AUTO": CHDIR "B:\2ND_WORD": CHDIR "A:"`

Sie befinden sich jetzt (vom Standardpfadnamen her) im Ordner "AUTO".

Ein Ordnername mit Rückwärts-Schrägstrich bedeutet, der entsprechende Ordner soll geöffnet werden. Zwei Punkte mit Rückwärts-Schrägstrich schließen den Ordner wieder:

`CHDIR "..\"` verläßt den Ordner AUTO-Ordner

Der eigentliche Dateiname besteht (genauso wie Ordnernamen) aus bis zu acht Zeichen, dann optional eine sog. Extension: ein Punkt und drei weitere Zeichen. Die Extension ist normalerweise:

- .BAS für BASIC-Programme
- .PRG, .TOS oder .TTP für Maschinenprogramme
- .DOC, TXT für Texte, Anleitungen

Einstellung der RS 232 - Parameter

Die Übertragungsparameter der RS232-Schnittstelle können Sie mit folgendem Programm setzen:

```
100 B19200=0:B9600=1:B4800=2:B3600=3:B2400=4:B2000=5:B1800=6
110 B1200=7:B600=8:B300=9:B200=10:B150=11:B134=12:B110=13:B75=14
120 B50=15:Kein_Handshake=0:Xon_Xoff=1:Rts_Cts=2:Beides=3
130 Even=6:Odd=4:None=0:Stop1=8:Stop1_Komma_5=16:Stop2=24
140 Bit8=0:Bit7=32:Bit6=64:Bit5=96
150 '
160 Baudrate=B300:Handshake=Rts_Cts:Parity=Even:Stopbits=Stop1
170 Datenbits=Bit8
180 '
190 Ucr=Parity+Stopbits+Datenbits
200 XBIOS(,15,Baudrate,Handshake,Ucr,-1,-1,-1)
210 OPEN "V24",1
```

Steuerzeichen des VT 52 - Standards

CHR$(7) Gibt einen Ton aus

CHR$(8) Cursor eins nach links

CHR$(9) Cursor vor zur nächsten 8-er-Spalte (TAB)

CHR$(10) Cursor nach unten (Zeilenvorschub) (auch: CHR$(11), CHR$(12))

CHR$(13) Cursor an linken Rand (Carriage Return)

CHR$ (27)+"X" wird im folgenden ESC X genannt. Bei den folgenden Steuerzeichenkombinationen handelt es sich um die sogenannten "ESCape-Sequenzen":

ESC A	bewegt den Cursor um eine Zeile nach oben
ESC B	bewegt den Cursor um eine Zeile nach unten
ESC C	bewegt den Cursor um eine Spalte nach rechts
ESC D	bewegt den Cursor um eine Spalte nach links
ESC E	löscht den Bildschirm und stellt den Cursor in die linke obere Ecke
ESC H	stellt den Cursor in die linke obere Ecke
ESC I	bewegt den Cursor um eine Zeile nach oben und schiebt, wenn nötig, den Bildschirm um eine Zeile nach unten
ESC J	Rest des Bildschirms löschen
ESC K	Rest der Zeile löschen
ESC L	Zeile einfügen
ESC M	Zeile herauslöschen
ESC Y n1 n2	Cursor positionieren; n1 = CHR$(Zeile+32), n2 = CHR$(Spalte+32).
ESC b n	Schriftfarbe setzen; n ist CHR$(Farbe).
ESC c n	Hintergrundfarbe setzen; n ist CHR$(Farbe).
ESC d	Bildschirm bis zum Cursor löschen
ESC e	Cursor einschalten
ESC f	Cursor ausschalten
ESC j	Cursorposition speichern
ESC k	gespeicherte Cursorposition abrufen
ESC l	Zeile löschen
ESC o	Zeile bis zum Cursor löschen
ESC p	Inversdarstellung ein
ESC q	Inversdarstellung aus
ESC v	Automatischer Überlauf ein (Cursor rückt nach 80. Spalte vor in die nächste Zeile.)
ESC w	Automatischer Überlauf aus (Cursor bleibt an 79. Spalte stehen.)

Die TOS-Fehlermeldungen

Fehler-Nr.	*Bedeutung*
0	Alles in Ordnung, Operation ok.
1	allgmeiner Fehler
2	Laufwerk nicht bereit
3	Befehl nicht bekannt
4	Prüfsummenfehler
5	Nicht ausführbar
6	Spur nicht gefunden/Daten defekt/Disk defekt
7	Bootsektor nicht ok
8	Sektor nicht gefunden/Daten defekt
9	Kein Papier mehr
10	Schreibfehler (Daten defekt)
11	Lesefehler (Daten defekt)
12	allgemeiner Fehler
13	Disk ist schreibgeschützt
14	Disk wurde gewechselt
15	Diskettenformat nicht bekannt
16	Sektoren falsch geschrieben ("Verify Error")
17	Keine Disk im Laufwerk
18-31	<nicht belegt>
32	Funktionsnummer ungültig
33	(Programm-) Datei nicht gefunden
34	Pfad nicht gefunden
35	zuviele Dateien offen - eröffnen neuer nicht mehr möglich
36	Zugriff unmöglich/verboten
37	Dateikennziffer ("Handle") ungültig
38	<nicht belegt>
39	nicht genug Speicher übrig
40	Kein Speicherblock an dieser Adresse
41-45	<nicht belegt>
46	Laufwerksbezeichnung ungültig
47	<nicht belegt>
48	Umbenennen geht nur mit gleichem Laufwerk

Fehler-Nr.	*Bedeutung*
49	keine weitere Datei vorhanden
50-57	<nicht belegt>
58	Record ist geschützt
59	Passender Schutz nicht gefunden
60-63	<nicht belegt>
64	Bereichsfehler
65	Interner GEMDOS-Fehler
66	Ungültiges Programmdateiformat
67	Fehler bei Mshrink/Speicherblock nicht vergrößerbar

Tabelle der Fehlermeldungen

1 Structure too long

Mehr als 64k Programmtext befinden sich zwischen zwei Strukturwörtern (z.B. FOR...NEXT, WHILE..WEND, REPEAT..UNTIL)

2 Syntax error

Irgend etwas stimmt an dem Befehl nicht, den Sie gerade eingetippt haben (z.B. Tippfehler). Dabei haben Sie jedoch nicht eine Unmöglichkeit verlangt, wie z.B. eine Zahl durch null zu teilen, sondern OMIKRON.BASIC kann den Befehl an sich nicht verstehen.

3 RETURN without GOSUB

Das Programm ist auf den Befehl RETURN gestoßen, ohne daß es sich in einem mit GOSUB aufgerufenen Unterprogramm befand. Dies passiert z.B. dann, wenn Sie Ihre Unterprogrammdefinitionen hinter das Hauptprogramm geschrieben haben und Ihr Hauptprogramm nicht mit END o.ä. endet.

Beispiel:

```
100 GOSUB Eingabe
110 PRINT "dreimal";A;"ist";3*A
500-Eingabe
510 INPUT"Nenne mir eine Zahl";A
520 RETURN
```

Dieses Programm läuft nun zuerst einmal korrekt ab, anschließend taucht aber ein ?RETURN without GOSUB in 520 auf. Denn das Programm ist einfach in Ihr Unterprogramm "reingelaufen". Fügen Sie ein:

```
499 END
```

Ein anderer häufiger Fehler besteht darin, daß eine Schleife nicht ordnungsgemäß verlassen wurde:

```
500 FOR I=1 TO 100
510 INPUT "ARTIKEL"+STR$(I)+": ";A$(I):IF A$(I)="" THEN GOTO
530
520 NEXT I
530 RETURN
```

Schleifen bricht man korrekterweise gar nicht oder nur mit EXIT vorzeitig ab.

4 Out of DATA

Ein READ-Befehl findet keine Daten mehr. Abhilfe: Mehr Daten eingeben, Überprüfung auf Daten-Ende in das Programm einbauen oder nochmal abzählen.

5 Illegal function call

Eine Funktion oder ein Befehl wurde auf eine Art und Weise verwendet, auf die sie nicht verwendet werden darf. Abhilfe: die Argumente überprüfen.

6 Overflow

Der Rechenbereich von OMIKRON.BASIC geht bis 5.11E+4931 (5,11 mal 10 hoch 4931). Wenn Sie diesen Rechenbereich überschreiten (z.B. durch PRINT 2^10'0000), wird die ?Overflow-Fehlermeldung ausgegeben.

7 Out of memory

Zwei Bedeutungen:

1. kein Speicherplatz mehr. Abhilfe: Variablen-Felder kleiner dimensionieren, ohne RAM-Disk booten, GEMDOS-Speicher verkleinern (CLEAR-Befehl).

2. kein Platz mehr auf dem Prozessor-Stapel. Platz auf dem Prozessor-Stapel wird belegt durch:

- offene Schleifen
- aufgerufene Unterprogramme
- aufgerufene Prozeduren oder Funktionen
- bestimmte Befehle wie SORT

Oft tritt dieser Fehler auf, wenn man mit GOSUB in ein Unterprogramm springt und dann mit GOTO irgendwo hin springt, ohne daß ein RETURN erfolgt ist, beispielweise:

```
10 GOSUB 20
20 GOTO 10
```

Abhilfe: Programm auf oben genannten Fehler überprüfen. Falls der Fehler durch zu tiefe Rekursion (Selbstaufruf von Funktionen oder Prozeduren) entstand, können Sie den Stapelplatz mit dem Befehl CLEAR vergrößern (siehe dort).

8 Undefined Statement

Das gewünschte Sprungziel existiert nicht. Beispiele:

- GOTO 100 - Zeilennummer 100 gibt's nicht
- Datei_Ausgeben(3,4) - Prozedur Datei_Ausgeben existiert nicht bzw. existiert, braucht aber eine andere Anzahl von Parametern
- LIUST (vertippt, sollte list heißen) - Prozedur Liust existiert nicht

9 Subscript out of range

Sie verwenden ein Element eines Variablen-Feldes, dessen Feldindex größer ist, als das Feld dimensioniert ist.

Beispiel:

```
10 DIM A$(100)
20 I=200:PRINT A$(I)
```

Häufig tritt dieser Fehler auch auf, wenn Sie ein Feld nicht selbst definiert haben, sondern es automatisch von OMIKRON.BASIC dimensioniert wurde. Siehe Befehl DIM.

10 Duplicate definition

Sie definieren z.B. eine Prozedur/Funktion zweimal.

Beispiel:
```
1000 DEF PROC A(X)
1010 PRINT CHR$(27);CHR$(X):RETURN
2000 DEF PROC A(X)
2010 PRINT X;" zum Quadrat ist ";X^2:RETURN
```

Abhilfe: Vernünftige Namen verwenden, dann passiert so etwas nicht (...so leicht).

11 Division by zero

Es wurde versucht, durch null zu teilen.

12 Illegal direct

Der gewünschte Befehl darf nicht im Direktmodus ausgeführt werden.

13 Type mismatch

Falscher Variablentyp. Beispiele:

A="23"	- Einer Zahlenvariablen darf kein Text zugewiesen werden
B$=23	- Einer Textvariablen darf keine Zahl zugewiesen werden
PRINT 3+"4"	- in Berechnungen nicht Zahlen und Texte mischen!
Q=B(X$)	- Als Feldindex sind keine Textvariablen zugelassen

14 RETURN without function

Entspricht ?RETURN without GOSUB, jedoch ist das Programm hier nicht auf ein gewöhnliches RETURN, sondern auf ein RETURN <Wert> (wie es in mehreren Funktionen verwendet wird, siehe dort) gestoßen.

15 String too long

Die Länge von Zeichenketten (Strings) ist auf 32766 Zeichen begrenzt.

16 Formula too complex

Zu komplizierte Berechnung, die den verfügbaren Stackplatz sprengt. Abhilfe: Formel vereinfachen, in mehrere Teilberechnungen aufteilen, oder mit CLEAR (siehe dort) mehr Prozessor-Stapelplatz reservieren.

17 Can't continue

Nach Fehlermeldungen oder nachdem Programmzeilen verändert wurden, ist kein CONT mehr möglich.

18 Undefined user function

Die gewünschte Funktion wurde nicht definiert.

19 No RESUME

Das Programm trifft in einer mit ON ERROR GOTO aufgerufenen Fehlerbehandlungs-Routine auf den END-Befehl oder auf das Programmende. Abhilfe: RESUME einbauen oder den Fehler gezielt mit ON ERROR GOTO 0 ausgeben (siehe ON ERROR GOTO).

20 RESUME without error

Das Programm triff auf RESUME, obwohl es gerade nicht in einer mit ON ERROR GOTO aufgerufenen Fehlerbehandlungs-Routine ist. Das passiert hauptsächlich, indem (wie bei ?RETURN without GOSUB erklärt) das Programm einfach in die Fehler-Routine "hineinläuft".

21 use EXIT

In manchen BASIC-Dialekten kann man eine Schleife so verlassen (Zeile 120):

```
100 FOR I=1 TO 10
110     INPUT"Name";Name$(I)
120 IF Name$(I)<>"/" THEN NEXT
```

In OMIKRON.BASIC können Schleifen nur mit dem Befehl EXIT vorzeitig verlassen werden

22 Missing operand

Ein Operand fehlt. Beispiele:

1000 DATA 1,2,3, (nach Komma muß ein Wert folgen)

PRINT 3+ (nach Rechenzeichen muß ein Wert folgen)

23 Line buffer overflow

Eine Programmzeile darf bei der Eingabe höchstens 255 Zeichen, beim LISTen höchstens 512 Zeichen enthalten. Abhilfe: Zeile auftrennen in mehrere Zeilen.

Der Fehler tritt auch dann gerne auf, wenn das Programm mit "PROTECT.BAS" (auf der Demodisk) geschützt wurde oder wenn es teilweise durch Schreib- oder Lesefehler zerstört wurde.

24 REPEAT without UNTIL

Eine Schleife wurde mit REPEAT geöffnet, doch es gibt kein zugehöriges UNTIL.

25 UNTIL without REPEAT

Zu einem UNTIL fehlt das zugehörige REPEAT.

26 FOR without NEXT

Eine Schleife wurde mit FOR geöffnet, doch es gibt kein zugehöriges NEXT.

27 NEXT without FOR

Zu einem NEXT fehlt das zugehörige FOR.

28 IF without THEN or ENDIF

Zwei Möglichkeiten:

- Zu einem IF fehlt das THEN
- Zu einem mehrzeiligen IF fehlt das ENDIF,

Beispiel:

```
100 IF Anzahl>Max_Anzahl THEN
110     PRINT"Die Anzahl ist zu groß"
120     PRINT"Wählen Sie eine kleinere Anzahl"
130     GOTO Eingabe
```

Hier fehlt 140 ENDIF, um das mehrzeilige IF abzuschließen.

29 WHILE without WEND

Eine Schleife wurde mit WHILE geöffnet, doch es gibt kein zugehöriges WEND.

30 WEND without WHILE

Zu einem WEND fehlt das zugehörige WHILE.

31 THEN, ELSE or ENDIF without IF or THEN

Zu einem THEN, ELSE oder ENDIF fehlt das zugehörige IF bzw. zu einem ELSE oder ENDIF fehlt das zugehörige THEN.

32 <intern verwendet>

33 Reset

Sie haben die RESET-Taste hinten am Computer gedrückt. Ihr Programm bleibt bei RESET erhalten, jedoch wird ein CLEAR ausgeführt, die Variablen werden also gelöscht.

34 Bus error

Bus-Fehler in einem mit CALL oder USR aufgerufenen Maschinenprogramm.

35 Adress error

Adress-Fehler in einem mit CALL oder USR aufgerufenen Maschinenprogramm.

36 Unknown opcode

In einem mit CALL oder USR aufgerufenen Maschinenprogramm kommt ein unbekannter Maschinenbefehl vor.

37 Division by Zero

In einem mit CALL oder USR aufgerufenen Maschinenprogramm kommt eine Division durch Null vor.

38 - 43 <Nicht belegt>

44 Out of memory. SAVE or re-CLEAR immedeately

Dem BASIC-Editor bzw. dem OMIKRON.BASIC ist der interne Speicher ausgegangen. Wenn Sie jetzt nicht sofort Ihr Programm speichern, bzw. mittels CLEAR mehr Speicher anfordern, kann es zu einem Absturz kommen, der den Verlust Ihres Programmes mit sich bringt.

45 EXIT without structure

Der Befehl EXIT soll ausgeführt werden, obwohl keine Schleife offen und kein Unterprogramm aufgerufen ist.

46 Use EXIT TO in functions

Mehrzeilige Funktionen können nicht mit einfachem EXIT verlassen werden. Verwenden Sie stattdessen EXIT TO.

47 Not regular matrix

Die Matrix, die Sie mit MAT INV invertieren wollten, war nicht regulär, d.h., es existiert keine Inverse dazu.

48 <Nicht belegt>

49 Bad line number

Sie haben vor einer Zeile keine Zeilennummer geschrieben, obwohl Sie sich im Zeilennummer-Modus befinden. Vgl. auch MODE-Menü, Menüpunkt "PREFERENCES".

50 FIELD overflow

In einer FIELD-Anweisung wurden zuviel Daten angegeben.

51 <Intern verwendet>

52 Bad file number

Dateien (Files) dürfen nur Nummern von 1 bis 16 erhalten.

53 File not found

Die gewünschte Datei existiert nicht.

54 Bad file mode

Wenn Sie mit OPEN"I" geöffnete Dateien beschreiben oder mit OPEN"O" geöffnete lesen wollen, oder wenn Sie random-access- und sequentielle Dateien verwechseln, tritt dieser Fehler auf.

55 File already open

Eine bereits geöffnete Datei soll nochmals geöffnet werden. Dieser Fehler kann z.B. auftreten, wenn Sie aus Versehen versuchen, mehrere Dateien mit der gleichen Datei-Nummer zu öffnen, z.B.

```
100 OPEN "I",1,"Adressen.DAT"
110 OPEN "I",1,"Lager.DAT"
```

56 File not open

Eine Datei muß geöffnet werden, bevor man sie beschreiben oder aus ihr lesen kann.

57 TOS error #XX

Das Betriebssystem TOS meldet, daß Fehler Nr. XX aufgetreten ist.

Siehe Anhang TOS-Fehlermeldungen (Seite 196).

58 File already exists

Eine Datei gleichen Namens exisitiert bereits in dem angegebenen Verzeichnis. Bevor Sie eine neue Datei öffnen können, muß eine Datei gleichen Namens gelöscht werden. Der Fehler kann auch auftreten, wenn eine Datei schreibgeschützt ist und neu eröffnet werden soll. (Zum Bsp. bei SAVE)

59 File type mismatch

Eine Datei hatte den falschen Typ. Dieser Fehler kann auftreten, wenn Sie ein

Programm mittels LOAD einladen wollen, das kein BASIC-Programm ist, bzw. ein Programm mittels EXEC starten wollen, das nicht ausführbar ist.

60 Bad disk

Die eingelegte Diskette ist defekt. Verwenden Sie eine andere.

61 Disk full

Auf der Diskette oder Harddisk ist nicht mehr genügend Speicherplatz vorhanden, um das Programm oder die Datei darauf abzuspeichern. Den auf einer Diskette noch verfügbaren Speicherplatz erhalten Sie mit FRE.

62 Input past end

Es wurde versucht, über das Ende einer (sequentiellen) Datei, also über das EOF-Zeichen hinaus, Daten zu lesen. Abhilfe: Abfrage mit der EOF-Funktion vor der Eingabe.

63 Bad record number

Es wurde versucht, in einer random-access-Datei auf einen Record (Datensatznummer) zuzugreifen, der die Kapazität des GEMDOS sprengen würde.

64 Bad file name

In Datei-Namen sind einige Zeichen nicht zugelassen: Doppelpunkt, Komma, Semikolon, Punkt etc.

65 Path not found

Ein Verzeichnis oder eine Datei wurde nicht gefunden. Vgl. auch "File not found"

66 Direct statement in file

Ein Programm, das im ASCII-Format vorliegt und mit LOAD eingeladen werden soll, enthält Zeile(n), die keine Zeilennummer enthalten. Abhilfe: EDIT. Im Editor mit [Shift]-[F8] das Programm einladen, fehlende Zeilennumern einfügen.

67 Too many files

Im Inhaltsverzeichnis einer Diskette ist nur Platz für eine begrenzte Anzahl Einträge. Abhilfe: Programme/Dateien mit dem KILL-Befehl löschen.

68 Write error

Beim Schreiben von Daten auf Diskette/Festplatte ist ein Fehler aufgetreten. Vermutlich ist die Diskette defekt.

69 Read error

Beim Lesen der Daten von Diskette/Festplatte ist ein Fehler aufgetreten. Vermutlich ist die Diskette defekt.

70 Disk write protected

Die Diskette ist schreibgeschützt und Sie wollten etwas darauf abspeichern. Abhilfe: Schreibschutz entfernen.

71 Illegal pointer

Sie versuchen mit dem Operator "*" auf einen falschen oder nicht-existenten Pointer zuzugreifen. Vermutlich haben Sie die falsche Zeigervariable verwendet, oder diese noch nicht mittels &-Operator eingerichtet, z.B.

```
CLEAR
A=*T
? Illegal pointer
```

72 Illegal SELECT-CASE-Structur

Sie haben eine fehlerhafte SELECT-CASE-Struktur in Ihrem Programm. Wahrscheinlich fehlt das END_SELECT bzw. das DEFAULT oder Sie haben eine falsche Strukturtiefe. Abhilfe: Bei jeder neuen Struktur etwas einrücken; dadurch wird das Programm übersichtlicher und Sie sehen sofort, wo eine Struktur noch nicht korrekt geschlossen wurde.

BIOS - Funktionen

Im folgenden eine äußerst knappe Auflistung der Funktionen des BIOS (Basic Input / Output System). Dieser Betriebssystemteil hat zwar nichts mit OMIKRON.BASIC zu tun, dennoch haben wir diesen Teil auf Wunsch vieler Kunden in das Handbuch eingefügt. Genauere Informationen über die einzelnen Funktionen finden sie in entsprechender Fachliteratur.

BIOS(,0,L Zeiger) - getmpb

(Wird vom GEMDOS benötigt)

BIOS(R,1,Device) - bconstat

Stellt fest, ob das Gerät ein Zeichen gesendet hat. R=0: Kein Zeichen wurde gesendet, R=-1: Mindestens ein Zeichen wurde gesendet.

Device = 1 für RS232

Device = 2 für Tastatur

Device = 3 für MIDI

BIOS(R,2,Device) - bconin

Liest ein Zeichen vom jeweiligen Gerät ein. R: Bits 0..7: das eingelesene Zeichen

Für Device=2 enthalten die Bits 16..23 den Scancode der betreffenden Taste. Ist zusätzlich das entsprechende Bit in der Systemvariablen "conterm" gesetzt, dann findet man in den Bits 24..31 zusätzlich den aktuellen Shift-Status (vgl. BIOS(11)).

Device = 0 für Centronics-Port

Device = 1 für RS232

Device = 2 für Tastatur

Device = 3 für MIDI

BIOS(,3,Device,Zeichen) - bconout

Gibt das Zeichen an das jeweilige Gerät aus.

Achtung: Erst wenn das Zeichen geschrieben werden konnte, kehrt diese Routine zurück. Also aufpassen bei Druckern etc.!

Device = 0 für Centronics-Port

Device = 1 für RS232

Device = 2 für Bildschirm

Device = 3 für MIDI

Device = 4 für Tastaturprozessor

Device = 5 für Bildschirm ohne Terminalemulation

BIOS(R,4,rwflag,L Buffer,Sektorzahl,Sektornr,Laufwerk) - rwabs

Liest oder schreibt Sektoren auf beliebigem Laufwerk.

rwflag = 0 für lesen, Fehler bei Diskettenwechsel melden

rwflag = 1 für schreiben, Fehler bei Diskettenwechsel melden

rwflag = 2 für lesen, Diskettenwechsel ignorieren

rwflag = 3 für schreiben, Diskettenwechsel ignorieren

Buffer: Zeiger auf einen Speicherbereich der geschrieben werden soll bzw. von dem gelesen werden soll.

Sektorzahl: Anzahl der zu lesenden/zu schreibenden Sektoren.

Sektornr: Startsektor ab dem geschrieben/gelesen werden soll.

Laufwerk: 0=A:, 1=B:, ...

R=0: Kein Fehler, <0: Fehler aufgetreten.

BIOS(R,5,Vektornummer,L Vektoradresse) - setexc

Setzt den Exeption-Vektor. Ist die Adresse -1, so wird nicht gesetzt, sondern nur abgefragt

Vektornummer: Nummer des Vektors

Vektoradresse: Neue Adresse oder -1.

BIOS(R,6) - tickcal

Ergibt die Anzahl Millisekunden zwischen zwei Timeraufrufen

BIOS(R,7,Laufwerk) - getbpb

Ermittelt die Adresse des BIOS-Parameter-Blocks des Laufwerks. Der Block besteht aus neun Integer-Wort-Zahlen:

Bytes je Sektor, Sektoren je Cluster, Bytes je Cluster, Sektoren des Directory, Sektoren je FAT, Sektornr. des 2. FAT, Startsektornr. der Daten, Anzahl Daten-Cluster, (intern verwendet)

BIOS(R,8,Device) - bcostat

Stellt fest, ob das Gerät bereit ist, ein Zeichen zu empfangen

Device = 0 für Centronics-Port

Device = 1 für RS232

Device = 3 für MIDI

Device = 4 für Tastaturprozessor

R=0: Gerät nicht bereit, R=-1: Gerät bereit.

BIOS(R,9,Laufwerk) - mediach

Stellt fest, ob Diskette gewechselt wurde: R=0: nein, R=1: vielleicht,

R=2: ja.

BIOS(R,10) - drvmap

Ermittelt logische Laufwerke: BIT(0,R) für Laufwerk A:, BIT(1,R) für B: ...

BIOS(R,11,Status) - kbshift

Setzt Status der SHIFT-Tasten; bei Status=-1 wird nur abgefragt.

BIT(0,R) bzw. BIT(0,Status) für rechtes SHIFT gedrückt

BIT(1,R) bzw. BIT(1,Status) für linkes SHIFT gedrückt

BIT(2,R) bzw. BIT(2,Status) für CONTROL gedrückt

BIT(3,R) bzw. BIT(3,Status) für ALTERNATE gedrückt

BIT(4,R) bzw. BIT(4,Status) für CAPS LOCK aktiv

XBIOS - Funktionen

Im folgenden eine äußerst knappe Auflistung der Funktionen des XBIOS (eXtended Basic Input / Output System). Dieser Betriebssystemteil hat ebenfalls nichts mit OMIKRON.BASIC zu tun. Genauere Informationen über die einzelnen Funktionen finden sie in entsprechender Fachliteratur.

XBIOS(,0,Modus,L Param,L Routine) - initmous

Initialisiert die Maus.

Modus: 0= Maus ausschalten

1= Maus einschalten, relativer Modus

2= Maus einschalten, absoluter Modus

3= (unbenutzt)

4= Maus einschalten, Tastaturemulation

Param ist ein Zeiger auf:

.B 0= Y-Achse zählt von unten nach oben, 1= von oben nach unten

.B für Maustasten: Bit 0: bei drücken Mausposition melden

Bit 1: bei loslassen Mausposition melden

Bit 2: bei Drücken Tastencodes melden

.B x-Teilung der Bewegung (nur im relativ-Modus benutzt)

.B y-Teilung der Bewegung (nur im relativ-Modus benutzt)

.B xmax (nur im absolut-Modus benutzt)

.B ymay "

.B xstart "

.B ystart "

Routine ist die Adresse der Maus-Routine

XBIOS(R,1,Speichermenge) - ssbrk

(Muß vor der Initialisierung des Betriebssystems aufgerufen werden)

XBIOS(R,2) - physbase

Ermittelt die Adresse, von der der Videoprozessor den Bildschirm darstellt.

XBIOS(R,3) - logbase

Ermittelt die Adresse des Bildschirms, auf den gerade ausgegeben wird.

XBIOS(R,4) - getrez

Ermittelt die Bildschirmauflösung.

0 = 320*200 Punkte, 16 Farben (Farbmonitor)

1 = 640*200 Punkte, 4 Farben (Farbmonitor)

2 = 640*400 Punkte, 2 Farben (s/w-Monitor)

XBIOS(,5,L Adr1,L Adr2,Auflösung) - setscreen

Setzt Ausgabe-Adresse(Adr1), Anzeige-Adresse(Adr2) und Auflösung (s.o.).

Eine -1 als Wert sorgt dafür, daß der entsprechende Wert nicht gesetzt wird.

XBIOS(,6,L Zeiger) - setpalette

Setzt Farben (siehe PALETTE-Befehl). Der Zeiger zeigt auf einen Speicherbereich, der 16 Integer-Wort-Werte mit den Farben enthält.

XBIOS(R,7,Farbnummer,Farbwert) - setcolor

Setzt eine Farbe. Mit einem Farbwert von -1 wird nur abgefragt.

XBIOS(R,8,L Buf,0,0,Laufwerk,Sektor,Spur,Seite,Anzahl) - floprd

Liest Sektoren von Laufwerk A:(0) oder B:(1). Buf ist ein Zeiger auf den Buffer, der Sektor geht von 1 bis 9 oder 10, die Spur von 0 bis 79 oder mehr, die Seite ist 0 oder 1 (bei 350-Kbyte-Laufwerken nur 0). Über das Spurende kann nicht hinausgelesen werden.

XBIOS(R,9,L Buf,0,0,Laufwerk,Sektor,Spur,Seite,Anzahl) - flopwr

Schreibt Sektoren. Parameter wie floprd.

XBIOS(R,10,L Buf,L 0,Laufwerk,Sektoranzahl,Spur,Seite,Interleave,

L $87654321,Init) - flopfmt

Formatiert eine Spur. Buf zeigt auf 10 Kbyte Platz, die Sektoranzahl ist 9 oder 10, Interleave ist der interleave factor (nehmen sie Interleave=1!), Init sind die Daten, die die Spur anfangs enthalten soll. Keines der beiden Bytes der Zahl Init darf einen Wert von $F0 bis $FF haben. Nichterklärte Parameter siehe floprd.

XBIOS(R,11) - getdsb

Die "einzige" Möglichkeit, die Variable R zuverlässig auf null zu setzen

(O Ironie, verlass mich nie!)

XBIOS(,12,Länge,L Pointer) - midiws

Schreibt Daten auf dem MIDI-Kanal hinaus. Pointer ist ein Zeiger auf die Daten im Speicher.

XBIOS(,13,Nummer,L Adresse) - mfpint

Installiert eine Interruptroutine für Nummer:

0=Centronics Busy, 1=RS232 DCD, 2=RS232 CTS, 4=Timer D, 5=Timer C,

6=Tastatur- und MIDI-ACIA's, 7=FDC- und DMA-Chips, 8=Timer B,

9=RS232 Sendefehler, 10=RS232-Sendebuffer leer, 11=RS232 Empfangsfehler,

12=RS232-Empfangsbuffer voll, 13=Timer A, 14=RS232 Ring Indicator,

15=Monochrom-Monitor detect

XBIOS(R,14,Device) - iorec

Ermittelt den Zeiger auf folgende Daten:

.L Bufferadresse

.W Buffergröße

.W Offset, wo neue Daten hinzukommen (Head)

.W Offset, wo Daten herauszunehmen sind (Tail)

.W Low water mark

.W High water mark

bei folgenden Geräten:

Device = 0 für RS232 - ermittelt Eingabebuffer; Ausgabebuffer schließt an

Device = 1 für Tastatur

Device = 2 für MIDI

XBIOS(,15,Baud,Ctrl,Ucr,Rsr,Tsr,Scr) - rsconf

Baud ist ein Wert von 0-15 (oder -1 zum nicht-setzen). Den Werten von 0-15 sind der Reihenfolge nach folgende Baudraten zugeordnet:

19200,9600,4800,3600,2400,2000,1800,1200,600,300,200,150,134,110,75,50

Ctrl: 0=kein Handshake, 1=XON/XOFF, 2=RTS/CTS, 3=beides, -1=nicht setzen

Ucr, Rsr, Tsr, Scr sind Eingaben, um die gleichnamigen Register des MFP zu setzen. Wird -1 angegeben, so wird das entsprechende Register nicht gesetzt.

UCR Bit 0: unbenutzt

Bit 1: 0=odd parity, 1=even parity

Bit 2: 0=no parity, 1=parity

Bit 3,4: 0=synchron, 1=1 Stopbit, 2=1.5 Stopbits, 3=2 Stopbits

Bit 5,6: 0=8 Bits, 1=7 Bits, 2=6 Bits, 3=5 Bits

Bit 7: 0=Frequenz nicht teilen (nur synchron), 1=Frequenz teilen

RSR Bit 0: 1=RS232-Empfänger ein, 0=aus

Bit 1: SCR-Zeichen mit übertragen oder nicht (nur synchron)

Bit 2-7 sind nur abfragbar, nicht setzbar

TSR Bit 0: 1=RS232-Sender ein, 0=aus

Bit 1,2: 0=Ausgang hochohmig, 1=H, 2=L, 3=Ausgang auf Eingang

Bit 3: 1=Break senden (nur asynchron)

Bit 5: 1=Empfänger einschalten, wenn Zeichen fertig gesendet

Bit 4,6,7: nicht setzbar

SCR Enthält Synchronisationsbyte für synchron-Betrieb

XBIOS(R,16,L Tab1,L Tab2,L Tab3) - keytbl

Setzt die Tastaturtabellen. Tab1, Tab2 und Tab3 sind Zeiger auf Tastaturtabellen für normal, mit SHIFT und für CAPS LOCK (oder -1 für nichtsetzen). R ist dann ein Zeiger auf drei Zeiger auf die Tabellen.

XBIOS(R,17) - random

Ergibt eine 24-Bit-Zufallszahl

XBIOS(,18,L Buf,L Serial,Disktyp,Exec) - protobt

Erzeugt oder ändert einen Bootsektor.

Buf ist die Speicheradresse, an der der Bootsektor steht

Serial ist die neue 24-Bit-Seriennummer der Disk oder -1

Disktyp: 0= ST SS, 1= ST DS, 2= DT SS (360 K), 3= DT DS (720 K) oder -1

Exec: 0=Disk bootet nicht, 1=Disk bootet oder -1

XBIOS(R,19,L Buf,0,0,Laufwerk,Sektor,Spur,Seite,Anzahl) - flopver

Prüft Sektoren auf Lesbarkeit. Parameter wie floprd.

XBIOS(,20) - scrdmp

Erstellt Hardcopy (siehe HCOPY-Befehl)

XBIOS(R,21,Funktion,Geschwindigkeit) - cursconf

Bestimmt den Corsor.

Funktion: 0=Cursor aus, 1=Cursor ein, 2=Cursor blinkend, 3=Cursor stabil, 4=Blinkgeschwindigkeit setzen, 5=Blinkgeschwindigkeit abfragen

XBIOS(,22,HIGH(Zeit),LOW(Zeit)) - settime

Setzt Uhrzeit und Datum.

Bits 0- 4:Sekunden geteilt durch 2 (0-29; 0=0s, 1=2s, 2=4s, 3=6s...)

Bits 5-10:Minuten (0-59)

Bits 11-15:Stunden (0-23)

Bits 16-20:Tag (1-31)

Bits 21-24:Monat (1-12)

Bits 25-31:Jahr (0-119; 0=1980, 1=1981...)

XBIOS(R,23) - gettime

Ergibt Uhrzeit und Datum im obigen Format.

XBIOS(,24) - bioskeys

Setzt Änderungen der Tastaturtabelle über keytbl zurück.

XBIOS(,25,Länge,L Pointer) - ikbdws

Übergibt Daten an den Tastaturprozessor. Pointer ist ein Zeiger auf die Daten im Speicher.

XBIOS(,26,Nummer) - jdisint

Sperrt einen Interrupt des MFP. Siehe mfpint

XBIOS(,27,Nummer) - jenabint

Gibt einen Interrupt des MFP frei. Siehe mfpint

XBIOS(R,28,Daten,Registernummer) - giaccess

Liest / schreibt ein Register des Soundchips. Registernr Bit 7 =1: schreiben

XBIOS(,29,Bitnummer) - offgibit

Löscht ein Bit im Port A des Soundchips

XBIOS(,30,Bitnummer) - ongibit

Setzt ein Bit im Port A des Soundchips

XBIOS(,31,Timernummer,Control,Data,L Adresse) - xbtimer

Startet einen MFP-Timer. Nummer 0-3= Timer A-D, Adresse= Interruptroutine.

Control: 0 =Timer aus

1-7 =Vorteiler teilt durch 4/10/16/50/64/100/200

8 =Event Count Mode (nur Timer A,B)

9-15=Pulsweiten-Mode, Vorteiler 4/10/16/50/64/100/200 (nur A,B)

Für Timer C ist der Control-Wert mit 16 zu multiplizieren.

Data: Wert, auf den der Timer gesetzt werden soll, wenn er abgelaufen ist

XBIOS(,32,L Zeiger) - dosound

Spielt eine vorgegebene Klangfolge ab.

Datenbytes: 0-15 & Wert: Register 0-15 des Soundchips setzen

128 & Startwert: Setzt Startwert für Kommando 129

129 & 0-15 & Inc Addiert Inc zum Startwert, schreibt ihn

& Endwert ins Soundregister (0-15) und wiederholt dies, falls der Endwert noch nicht erreicht wurde

255 & Delay Wartet n/50 Sekunden

255 & 0 Ende

XBIOS(,33,Einstellung) - setprt

Stellt Daten für den Drucker ein:

Bit 0: 0=Matrix-, 1=Typenraddrucker

Bit 1: 0=Farb-, 1=s/w-Drucker

Bit 2: 0=ATARI-, 1=EPSON-Drucker

Bit 3: 0=Test-, 1=Quality-Modus

Bit 4: 0=Centronics, 1=RS232

Bit 5: 0=Endlos, 1=Einzelblatt

XBIOS(R,34) - kbdvbase

Ergibt einen Zeiger auf eine Tabelle mit folgenden Zeigern:

MIDI in, Keyboard-Fehler, MIDI-Fehler, IKBD Status, Maus-Routinen,

Uhrzeit-Routine, Joystick-Routinen.

XBIOS(R,35,Verzögerung,Wiederholgeschwindigkeit) - kbrate

Stellt den Tastatur-Repeat ein. Ein Wert von -1 bedeutet nicht setzen.

R enthält: Unteres Byte: Geschwindigkeit, oberes Byte: Verzögerung.

XBIOS(,36,L Pointer) - prtblk

Nochmals eine Hardcopy. Pointer zeigt dabei auf eine Tabelle mit sehr vielen Angaben.

XBIOS(,37) - vsync

Wartet auf vertical blanc (vertikalen Bildrücklauf)

XBIOS(,38,L Adresse) - supexec

Führt eine Maschinenroutine im Supervisor-Modus auf.

XBIOS(,39) - puntaes

Löscht bei älteren Betriebssystemen (solche, die von Disk geladen werden) das AES (muß aus dem AUTO-Ordner heraus aufgerufen werden).

GEMDOS - Funktionen

Im folgenden eine äußerst knappe Auflistung der Funktionen des GEMDOS (Graphics Environment Manager Disk Operating System). Dieser Betriebssystem- teil hat auch nichts mit dem OMIKRON.BASIC zu tun. Genauere Informationen über die einzelnen Funktionen finden sie in entsprechender Fachliteratur.

GEMDOS(,0) - pterm0

Beendet das momentan laufende Programm. Dies ist im Interpreter nicht Ihr BASIC-Programm, sondern der Interpreter selbst!

GEMDOS(R,1) - cconin

Holt ein Zeichen von der Tastatur und gibt es auf den Bildschirm aus.

GEMDOS(R,2,Zeichen) - cconout

Gibt ein Zeichen auf dem Bildschirm aus.

GEMDOS(R,3) - cauxin

Liest ein Zeichen über die RS232-Schnittstelle ein.

GEMDOS(,4,Zeichen) - cauxout

Gibt ein Zeichen über RS232 aus.

GEMDOS(,5,Zeichen) - cprnout

Gibt ein Zeichen an den Drucker aus.

GEMDOS(R,6,Zeichen) - crawio

Gibt ein Zeichen auf dem Bildschirm aus; ist der Wert des Zeichens 255, so wird ein Tastenwert wie bei INKEY$ zurückgegeben - oder null, wenn keine Taste gedrückt worden war.

GEMDOS(R,7) - crawcin

Holt ein Zeichen von der Tastatur. GEMDOS(R,8) - cnecin

Holt ein Zeichen von der Tastatur.

GEMDOS(,9,L Pointer) - cconws

Gibt ab dem Pointer aus dem Speicher Zeichen aus, bis ein Nullbyte erreicht wird.

GEMDOS(R,10,L Buffer) - cconrs

Läßt den Benutzer eine Zeile eingeben. Buffer zeigt zwei Bytes vor den eigentlichen Buffer für die Eingabe. Die zwei Zeichen davor sind: Maximale Länge; Tatsächliche Länge der Eingabe.

GEMDOS(R,11) - cconis

Testet ab, ob eine Taste gedrückt wurde.

R=0: Keine Taste gedrückt, -1: Taste gedrückt.

GEMDOS(R,14,Laufwerk) - dsetdrv

Stellt auf ein anderes Standardlaufwerk um und gibt die alte Laufwerknum- mer zurück. (0=A:, 1=B:...)

GEMDOS(R,16) - cconos

Die einzig sichere Methode, die Variable R auf den Wert $0000FFFF zu setzen.

(Hier wird abgetestet, ob der Bildschirm bereit ist, Daten zu empfangen...)

GEMDOS(R,17) - cprnos

Testet, ob der Drucker bereit ist, Daten zu empfangen.

GEMDOS(R,18) - cauxis

Testet, ob ein Zeichen von der RS232-Schnittstelle anliegt.

GEMDOS(R,19) - cauxos

Testet, ob ein Zeichen über RS232 gesendet werden kann.

GEMDOS(R,25) - dgetdrv

Ermittelt die Laufwerknummer des Standardlaufwerks.

GEMDOS(,26,L Adresse) - fsetdta

Setzt die Adresse des Buffers für sfirst und snext.

GEMDOS(R,32,L Wert) - super

Wert=0: Schaltet den Supervisor-Modus ein. R enthält den alten USP

Wert<>0: Schaltet den Supervisor-Modus aus. als Wert muß der alte USP übergeben werden, den man beim Einschalten des Supervisor-Modus mitgeteilt bekam.

Sie können unter OMIKRON.BASIC den Supervisor-Modus leider nicht einschalten: er ist es bereits. Verwenden Sie also diese Funktion niemals unter OMIKRON.BASIC!

GEMDOS(R,42) - tgetdate

Ermittelt das Datum.

Bits 0- 4:Tag (1-31)

Bits 5- 8:Monat (1-12)

Bits 9-15:Jahr (0-119; 0=1980, 1=1981...)

GEMDOS(,43,Datum) - tsetdate

Stellt das Datum ein. Format siehe getdate.

GEMDOS(R,44) - tgettime

Ermittelt die Uhrzeit.

Bits 0- 4:Sekunden geteilt durch 2 (0-29; 0=0s, 1=2s, 2=4s, 3=6s...)

Bits 5-10:Minuten (0-59)

Bits 11-15:Stunden (0-23)

GEMDOS(,45,Uhrzeit) - tsettime

Stellt die Uhrzeit ein. Format siehe gettime.

GEMDOS(R,47) - fgetdta

Ermittelt die Adresse des Buffers für sfirst und snext.

GEMDOS(R,48) - sversion

Ermittelt die Versionsnummer des GEMDOS. $1323 bedeutet: 19.23.

GEMDOS(,49,L Größe,Fehlercode) - ptermres

Beendet das Programm, löscht es jedoch nicht aus dem Speicher. Als Größe wird die Größe des Programmes plus der benötigten Daten plus 256 angegeben. Nicht unter OMIKRON.BASIC aufrufen!

GEMDOS(R,54,L Buffer,Laufwerknummer+1) - dfree

Füllt den Buffer mit folgenden Daten:

.L Freie Daten-Cluster

.L Daten-Cluster gesamt

.L Bytes je Sektor

.L Sektoren je Cluster

Ist die Laufwerknummer -1 angegeben (+1 macht null), so wird die Berechnung für das Standardlaufwerk durchgeführt.

GEMDOS(R,57,L Ordnername) - dcreate

Richtet ein Subdirectory (Ordner) ein. Ordnername ist ein Zeiger auf den Ordnernamen, der durch ein Nullbyte abgeschlossen ist.

GEMDOS(R,58,L Ordnername) - ddelete

Löscht ein Subdirectory (Ordner). Ordnername ist ein Zeiger auf den Ordnernamen, der durch ein Nullbyte abgeschlossen ist.

GEMDOS(R,59,L Pfadname) - dsetpath

Ändert den Standardpfad eines Laufwerks. Ist im Pfadnamen keine Laufwerksbezeichnung vorhanden, so wird der Standardpfad des Standardlaufwerks geändert. Pfadname ist ein Zeiger auf den Pfadnamen, der durch ein Nullbyte abgeschlossen ist.

GEMDOS(R,60,L Dateiname,Dateiattribut) - fcreate

Legt eine neue Datei an. Dateiname ist ein Zeiger auf den Dateinamen, der durch ein Nullbyte abgeschlossen ist.

Dateiattribut besteht aus: 1=R/O, 2=hidden, 4=SYS-File, 8=Volume (, 16=Ordner)

R enthält entweder ein Dateihandle (>=6) oder eine Fehlernummer.

GEMDOS(R,61,L Dateiname,Modus) - fopen

Eröffnet eine bereits vorhandene Datei bei Modus 0 für Lesen, bei 1 für Schreiben, bei 2 für Lesen und Schreiben. Rest wie fcreate.

GEMDOS(R,62,Handle) - fclose

Schließt eine Datei. Handle ist die von fcreate oder fopen ermittelte Dateihandle-Nummer. R enthält null oder eine Fehlernummer.

GEMDOS(R,63,Handle,L Anzahl,L Buf) - fread

Anzahl ist die Anzahl der zu lesenden Bytes, Buf ist die Speicheradresse, wohin die Bytes gelesen werden sollen.

R enthält die Anzahl der gelesenen Bytes oder eine Fehlernummer.

GEMDOS(R,64,Handle,L Anzahl,L Buf) - fwrite

Anzahl ist die Anzahl der zu schreibenden Bytes, Buf ist die Speicheradresse, von der die zu schreibenden Bytes geholt werden können.

R enthält die Anzahl der geschriebenen Bytes oder eine Fehlernummer.

GEMDOS(R,65,L Dateiname) - fdelete

Löscht eine Datei. R enthält null oder eine Fehlernummer.

GEMDOS(R,66,L Anzahl,Handle,Modus) - fseek

Setzt den Schreib-/Lesezeiger einer Datei. Modus: 0=relativ zum Dateianfang, 1=relativ zum alten Zeiger, 2=relativ zum Dateiende.

R enthält eine Fehlernummer, wenn über die Dateienden hinausgegangen wird.

GEMDOS(R,67,L Dateiname,Modus,Dateattribut) - fattrib

Setzt (Modus=1) oder Ermittelt (Modus=0) das Dateiattribut.

GEMDOS(R,69,Device) - fdup

Öffnet Datei auf Tastatur/Bildschirm(1), RS232(2) oder Drucker(3). R: Handle

GEMDOS(R,70,Device,Handle) - fforce

Lenkt die Ausgabe des Gerätes (s. fdup)auf eine offene Datei um.

GEMDOS(,71,L Buf,Laufwerknummer+1) - dgetpath

Schreibt in den 64 Byte großen Buffer (der durch Buf angegeben wird), den Pfadnamen des angegebenen Laufwerks. Laufwerk -1 heißt Standardlaufwerk.

GEMDOS(R,72,L Anzahl) - malloc

Macht genau das gleiche wie die BASIC-Funktion MEMORY. Ausnahme: ***CLEAR löscht mit malloc angelegte Speicherblöcke nicht.***

R enthält die Speicherblockadresse oder die Anzahl der freien Bytes.

GEMDOS(R,73,L Speicherblockadresse) - mfree

Löscht einen von malloc angelegten Speicherblock.

GEMDOS(R,74,0,L Base,L Anzahl) - mshrink

Verkleinert die Speicherreservierung für ein Programm. Base ist die Adresse der Basepage, Anzahl ist die Anzahl der von Programm und Daten benötigten Bytes plus 256. Nicht unter OMIKRON.BASIC aufrufen!

GEMDOS(R,75,Modus,L Dateiname,L Commtail,L Environment) - pexec

Lädt (Modus=3) oder lädt und startet (Modus=0) ein Programm.

Dateiname ist ein Zeiger auf den Dateinamen, Commtail ein Zeiger auf das übergebene Kommando (seine Länge muß als Byte vorneangestellt sein.), Environment zeigt auf den Environmentstring: "PATH="+chr$(0)+"A:\"+chr$(0)+chr$(255)

R enthält wie üblich null oder eine Fehlernummer.

Wichtig: es muß sehr viel GEMDOS-Speicher frei sein. Beispielsweise:

CLEAR:CLEAR FRE(0)+memory(-1)-10000

GEMDOS(,76,Rückgabewert) - pterm

Beendet das laufende Programm (z.B. den Interpreter) und gibt einen Wert zurück.

GEMDOS(R,78,L Dateiname,Attribut) - fsfirst

Sucht den ersten Eintrag eines Inhaltsverzeichnisses. Parameter wie bei OPEN "F". Siehe Anhang "Dateitypen bei OPEN"

GEMDOS(R,79) - fsnext

Sucht den jeweils nächsten Eintrag (nach fsfirst)

GEMDOS(R,86,0,L Old,L New) - frename

Entspricht NAME...AS

GEMDOS(R,87,L Buf,Handle,Modus) - fdattime

Setzt (Modus=1) oder ermittelt (Modus=0) Datum und Uhrzeit einer geöffneten Datei. Buf zeigt auf 4 Bytes, die Datum und Uhrzeit enthalten.

Systemvariablen des ATARI ST

Im folgenden eine knappe Auflistung einiger Systemvariablen des ATARI ST. Dieser Betriebssystemteil hat auch nichts mit dem OMIKRON.BASIC zu tun. Genauere Informationen über die einzelnen Variablen finden sie in entsprechender Fachliteratur.

Abfragen: R= PEEK(<Adresse>)
Setzen: POKE <Adresse>,<Wert> bei (.B)
R=WPEEK(<Adresse>) WPOKE <Adresse>,<Wert> bei (.W)
R=LPEEK(<Adresse>) LPOKE <Adresse>,<Wert> bei (.L)

Adresse	Name	Bedeutung
.L $404	etv_critic.	Stellt Dialogboxen für Lesefehler dar. Durch Umsetzen auf: MOVE.L 4(SP),D0 | RTS kann dies abgeschaltet werden.
.L $426	resvalid.	Ein Inhalt von $31415926 macht den Reset-Vektor gültig.
.L $42A	resvector.	Reset-Vektor. Wird bei RESET evtl. angesprungen.
.L $42E	phystop.	Ende des Speichers im ST.
.L $432	_membot.	Beginn des für Programme zur Verfügung stehenden Speichers.
.L $436	_memtop.	Ende desselben.
.W $444	_fverify.	Ungleich null bedeutet, daß bei allen Floppy-Schreibzugriffen ein Prüflesen ausgeführt wird.
.B $44C	sshiftmd.	Bildschirmauflösung (siehe BIOS; getrez)
.L $44E	_v_bas_ad.	logische Bildschirmadr. (vgl. BIOS; setscreen)
.W $452	vblsem.	VBL-Interrupts zulassen (1) oder sperren (0).
.W $454	nvbls.	Anzahl der Zeiger in der VBL-Zeigertabelle
.L $456	_vblqueue.	Zeiger auf Zeigertabelle auf VBL-Routinen
.L $45A	colorptr.	Setzen Sie den Zeiger auf eine Farbpalette hier hinein. Nach dem nächsten VBL-Interrupt ist die Palette gesetzt und der Wert wieder null.
.L $45E	screenpt.	Dasgleiche wie colorptr für die Anzeigeadresse des Bildschirms. Funktioniert bei manchen Betriebssystemen nicht richtig.
.L $462	_vbclock.	Zählt die Anzahl der Ausgeführten VBL-Routinen ($452 =0 sperrt auch diesen Zähler.)
.L $466	_frclock.	Zählt die Anzahl der VBL-Interrupts.
.L $472	hdv_bpb.	Vektor für BIOS getbpb
.L $476	hdv_rw.	Vektor für BIOS rwabs
.L $47E	hdv_mediach.	Vektor für BIOS mediach

.B $484	conterm.	Bit 0: Tastaturklick ein/aus 1: Tastatur-Repeat ein/aus 2: Ton bei PRINT CHR$(7); ein/aus 3: Shift-Bits bei BIOS bconin bzw. INKEY$ ein/aus. (vgl. BIOS(11))
.W $4A6	_nflops.	Anzahl angeschlossener Laufwerke
.L $4BA	_hz_200.	Entspricht TIMER.
.L $4C2	_drvbits.	Bitvektor für logische Laufwerke (Bit 0=A:, 2=C:)
.L $4FE	exec_os.	Vektor zur initialisierung des GEM AES

Befehlsgruppen-Index

Stringfunktionen

Operatoren siehe Seite 37ff

Mathematische Funktionen

Zahlenkonvertierung, Zahlensysteme

Stringfunktionen

Operatoren: 41ff

Variablentypen, Variablenverwaltung

Strukturbefehle

Diskettenbefehle

Ein-/Ausgabebefehle

Systembefehle

Grafikbefehle

GEM-Befehle

Sound-Befehle

Multitasking

Fehlerbehandlung

Systemvariablen

Steuerbefehle

Stichwortverzeichnis

V:

W:

X:

Z: